Trick of the Light: A Game Engine for Exploring Novel Fog of War Mechanics

Zackery Mason

A project report submitted to the faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Master of Science in
Interactive Media and Game Development

[illegible]

Brian Moriarty, Project Chair

[illegible]

Dean O'Donnell, Reader

[illegible]

Dr. Gillian Smith, Reader

Abstract

Trick of the Light is an experiment in strategic game design based on imperfect information in a unique fog of war setting. A hybrid of real-time-strategy, role-playing-game and roguelike genres, the game challenges players to maintain an expansive base system without being able to see anything beyond their own limited vision radius. All units, allied or enemy, maintain private memories about what they have seen, and must directly exchange information to keep up to date. The player acts as commander, making decisions and giving orders while dealing with adversaries, sabotage and misinformation. Testing was done to see if the new concepts could be understood in-game and garner any interest for further development, which proved to be positive in both cases despite complaints related to having less direct control over allies.

Acknowledgements

Very special thanks to advisor Brian Moriarty. His support allowed this work of passion to become a thesis-level project, and his experience in game design and the video game industry as a whole was an immense boon that I am lucky to have barely tapped into.

Additional thanks to IMGD undergrad Dave Allen for providing audio and music, and also to the IMGD community as a whole, not only for playtesting the game but also for going above and beyond with your genuine interest and suggestions that indicate a desire to watch the game grow. A game is nothing without players, but a real developer appreciates supporters even more.

A final thanks to all of my friends and family involved with development long before this became any official project, especially my parents, who were avid supports of everything I attempt since the very first "Hello, world." Your willingness to feign interest as I babbled on about technical jargon has gone neither unnoticed nor unappreciated.

Contents

Figures

1. Introduction

Trick of the Light started as a high school garage project called *Gridworld*: a practice exercise that shamelessly imitated game mechanics from several existing genres. Its primary inspiration was anthill-simulators such as *Sim Ant* (1991), which emphasize indirect control over swarms of autonomous entities rather than hands-on micromanagement of individual units. (Maxis) A grid-based engine was created to support a simple, hands-free simulation of miners breaking down walls and carrying quarried rocks to an ore smelter. More features were introduced as the project developed, including combat between miners, a greater variety of resources to harvest, upgrades using collected materials, etc. Everything was displayed via text output, with no interaction from the player beyond pressing ‘play’ to start things up and watch the show.

Everything changed when spiders were added. Originally they were coded as simple hunters that could stun miners and drag them away. The problem was that the miners were able to see the spiders coming and flee, collapsing the simulation into an endless cycle of running and chasing. The first solution considered was making the spiders invisible so they could sneak up on their prey. This presented a problem: How should invisible entities be displayed to the player, if at all?

Until this point, the player possessed an all-seeing perspective of the game world, but was limited to watching events unfold. If the design evolved to incorporate the player into the world as an active participant, some model of limited vision needed to be developed. This would necessitate forethought about what kind of experience the game would eventually gravitate towards. While brainstorming designs on what could make the game unique and include a player in the current state of the world, inspiration came from imagining a common trope among strategy games: the sacrificial scout.

In conventional real-time strategy (RTS) games, all allied forces share map visibility with each other and the player, who oversees everything from an abstract, top-down point of view. A typical early tactic in such games is to send an expendable unit, usually a worker or “peon,” out into the unknown to search for the location of enemy bases. As they move, their findings are continuously transmitted to the player and allies via their “telepathic” connection, even if they are half a world away. By the time the scout discovers an enemy, they are usually so far from their home base that it is more cost-efficient for the player to let them remain in place as a sort of remote camera, monitoring local activity until they are eventually discovered and executed by enemy units.

The game-vision mechanic that enables this strategy is commonly known as *fog of war.* It is obviously not intended to be realistic. This doesn’t mean that there is anything wrong with it. Gamers have been enjoying RTS telepathy and sacrificing peons for decades. Rather than a problem to be solved, it was a concept to be explored. Scouts don’t need to come back to report their findings. But what if they did? What parts of fog of war would need to be adapted, removed, or replaced for something else to take its place? What would that something else be, and would it make the game more enjoyable? Would the result still be considered fog of war?

Trick of the Light has ever since been dedicated to the exploration of these questions, eventually leading to the development of a full-blown memory logistic system and independent, intelligent handling of each unit’s internal game-state. While the concepts themselves aren’t new, the scope of which they’re implemented is the key factor: vision is personalized to each unit, replacing allied telepathy with a model in which every individual keeps track of their own memories about what they’ve seen, and can only share information by direct interaction with other units. The player is subject to the same limitations. Instead of leading their troops from some omnipresent cloud in the sky, they can only know what they see for themselves, or what they can learn directly from others.

This implementation of limited vision and dependency on others for information escalated into a play experience demanding a constant need for intelligence reports, with a heightened sense of paranoia about what information is still up-to-date. This led to deeper thinking about how this new economy of information could be abused with sabotage, trickery and other malicious strategies.

Trick of the Light was in full development for years before it was proposed as a Master's thesis. Being able to concentrate on it as an academic project provided an opportunity to elevate the game to a playable state that introduces its core mechanics and test to see if the novel unshared vision and memory systems would be understood and appreciated by players familiar with conventional fog of war.

2. History of fog of war

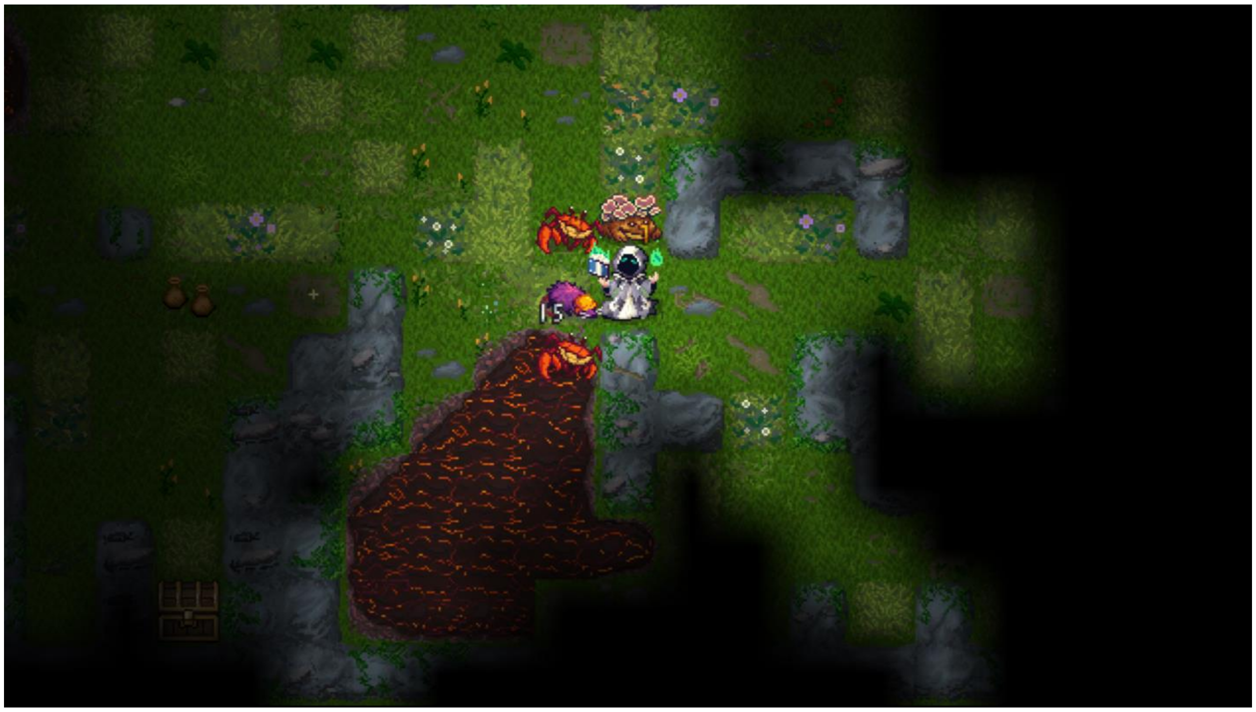


Figure 1. Example of vision in *Tangledeep* (2017), a roguelike game. Source: URL.

Fog of war is a term used to describe the mechanic of making only limited portions of a game map viewable, usually a combination of the areas immediately surrounding the player's character and all allied units (see Figure 1). Unit movement shifts these viewable zones and causes previously-visited areas to fade out of sight. This mechanic dynamically constrains the player's information, as areas outside their current viewing zones may contain active entities of interest. Progression requires eventual confrontation with whatever lies in the surrounding "fog," forcing players to think strategically about how to prepare for these unknowns.

The term fog of war is used by the military to describe the uncertainty of real-life combat situations. Command decisions are complicated by not being able to know exactly where the enemy is; intelligence may be unreliable or outdated, and information management is a stratagem critical to success. (Kiesling) Fog of war was often integrated into tabletop wargame simulations to capture this critical aspect of conflict. Implementations could range from only hiding the strength of enemy forces to making the terrain itself known only to a third-party referee until explored. (Setear)

Fog of war's use in tabletop games is limited by the fact that a referee is almost always needed to handle the distribution of information in a fair manner, as the physical instantiation of the game elements make it difficult for players to both hide their actions while ensuring every move conforms to the rules of conflict. (Guillory)

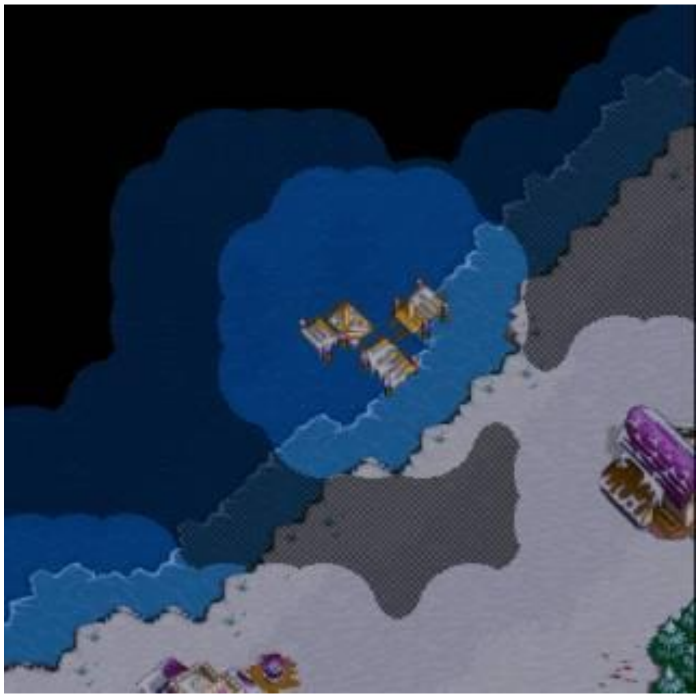


Figure 2. Example of vision in *Warcraft 2* (Blizzard, 1995), an RTS game. Source: URL.

The first *digital* game to incorporate the now-prevalent version of fog of war was Walter Bright's *Empire* in 1977. (Lewin) Due to limitations of the hardware, revealing an area made it permanently visible thereafter, even if the scouting unit left, but it still marks the first appearance of the concept of reducing the viewable area dynamically. Fog of war has since become a standard feature in multiple genres, including well-known examples from Blizzard's *Warcraft* (shown in Figure 2) and MicroProse's *Civilization* franchises, employed with little to no variation in the basic mechanics. (Wayward)

Fog of war games focus the player's attention within their viewable areas. Unseen territory is expected to be explored and conquered only after their objectives are completed in the currently visible zones. Once an area is under your control, it's usually considered "done," with little incentive for re-exploration if nothing is left behind. Even in unconquered territory, forward scouting always provides an accurate representation of the current state of obstacles or enemies the player may need to consider. Visual information is reliable: if the player can see something, they have no reason not to believe it isn't really there.

Fog of war is rarely the driving mechanic of a game, but it always bears a significant impact on a player's field of attention. In situations where the enemy's possibility space is completely known to the player (such as a multiuser game played against familiar opponents on a standard map), fog of war acts as a temporary shroud. Though it prevents direct observation of enemy activity, an experienced player can anticipate the likelihood of particular maneuvers and prepare accordingly. (Burgun, *Uncapped*)

However, when an enemy is unknown (typical in a single-player setting), fog of war imbues play with a sense of genuine mystery. Territory must still be explored and conquered inch by inch to achieve objectives, but the suspense of exploration is inherently rewarding. However, the replay value of revealed terrain is limited. Players can rapidly exhaust a map's secrets by deploying units widely; a completely revealed world loses the ambiguity that made it fun. (Burgun, *Fog of War*)

3. Game mechanics and their implementations

Zack Mason was the sole developer of the project from start to finish, though with plenty of advice from outside sources for difficult problems. This section goes into what the core mechanics of the game are, how they work and interact with each other, and the trials and tribulations that came with creating them.

3.1. Game Overview

Trick of the Light takes place in a 2D grid filled with units and / or items that occupy them. The game is turn-based, where units are capable of moving around and interacting with things world. Units can only see a limited distance around them due to an ever-present fog of war, but keep memories of the places they've been and the people / things they saw when they lose sight of them. Direct interaction between units allows them to share this information and keep up to date about the world-state. The game has factions of units working together, managing a base that necessitates logistics of supplies and information, with each unit acting independently completing tasks that benefit their team.

The player acts as a commander in charge of one of these groups, and is subject to the same limitations involving vision and memories. They're able to command allied units to do a variety of tasks but still lose track of them the moment they walk out of sight, requiring the results to be directly reported to them or discovered first hand. Gameplay takes place over different pre-generated levels, each with their own unique challenges and goals that require the played to learn and adapt to the mechanics presented to them.

3.2. Turn-based vs real-time

The decision to stick with a turn-based engine was not made lightly. As the concept was being finalized, there was much deliberation as to whether a real-time engine would be more appropriate for the intended style of play, and if so, whether it would be better to move the game to an existing engine for convenience, or make the extra effort required to create an optimized custom engine from scratch.

From a player’s perspective, real-time gameplay might seem to be the more exciting option. Games like *Total Annihilation* (Cavedog, 1997) and *Warcraft 3* (Blizzard, 2002) demonstrate how compelling a real-time, hero-centric adventure can be, providing a good mix of micro and macro management. There are constantly things to do at every given moment, demanding simultaneous focus on battles in progress while continuing unit production at the home base, to the point the challenge becomes trying to hand out as many commands as possible in as short a time frame the control scheme allows.

The main similarity *Trick of the Light* has to the RTS genre comes from the similar base and resource management model, but those systems will now be out of sight a vast majority of the time. Management comes from queueing up things to be created or built in advance and learning the results when they get reported later, with the AI handling the logistics of telling who to make what and bringing things where they need to be themselves. The high amount of actions required in an RTS aren’t as necessary when the things you can interact with are only within your view, and consequently have much greater weight. Determining what each unit’s long-term plan of action should be is better handled in a turn-based setting, where there is no time pressure to make rushed decisions that might result in bad outcomes.

The lack of complete vision over one's entire base at any given moment means that understanding updates involving it are essential. Interacting with a unit reporting in and learning everything they know at once can result in sudden upheavals to your understanding of the global map state. Such large-scale changes containing many potential subtleties are best pondered in a turn-based setting.

Similar considerations arose at almost every point of the imagined gameplay experience, implying the design of *Trick of the Light* favors a more contemplative experience than what a typical RTS is expected to deliver. It seemed wiser to allow players ample time to consider multiple strategies and make better-informed decisions rather than demand the fast-paced reactions a real-time engine necessitates.

3.2.1. 'Turn-based' energy system

The engine of *Trick of the Light* is 'turn based,' but not in the same way found in typical strategy games that use different phases for allies or enemies. Instead, it employs a tick-based energy system. Every game object that interacts with the world when it takes a 'turn' is a child of the Living class, hereafter referred to as a 'living' object. Such objects are assigned a 'next update' integer, put into a queue with every other living object and sorted so that the one with the smallest 'next update' number will be the next one activated. When a living object is activated, their update function is called, their 'next update' number is increased by their personal cooldown attribute and put back into the queue, usually behind almost every other object. The standard cooldown for most living objects is 1000 'ticks' (an arbitrary measure of in-game time). An object with a cooldown attribute of 500 updates twice as often as normal, while an object with a cooldown of 2000 would update at half the normal rate.

This tick-per-turn system added considerable freedom for controlling how often and in what order objects will update, but in practice it turned out there were very few cases of objects that needed to update at non-standard speeds. Faster or slower speeds only appear consistent when the cooldown attributes are even ratios of the standard 1000 ticks value. From a player's perspective, odd ratios such as 950 or 1050 seem to randomly give or take turns every few rounds. This led to most non-standard speeds being assigned to even ratios. Odd ratios were used in situations where their effect is hardly noticeable on a turn-to-turn basis, such as mining. Breaking down walls is a repetitive process involving dozens of attacks, most of which are done out of sight of a player, so raising or lowering the cooldown value per swing results in a way to control how much ore is collected over long periods of time in a way that's hardly noticeable to a normal player. Other similar situations arise, but in most cases a normal player won't realize the tick system is in place at all and assume a normal turn-based one, which isn't a problem.

3.2.2. Cooldown vs timer

Figure 3. *Dwarf Fortress* (Bay 12 Games, 2006), a popular simulation game and one of the primary inspirations for *Trick of the Light*, may appear to be turn-based but actually uses a cooldown system similar to that described below. Units can speed up or slow down doing activities like running or resting, making them take more or less turns over time. Source: URL.

This tick-based system described above was initially based off a cooldown-based system seen rarely in a select few roguelikes or simulations such as *Dwarf Fortress* (see Figure 3). In the old system, every turn reset a living object’s ‘next turn’ counter to its default instead of adding on to its existing value. When the next living object took its turn, every other living object in the queue would have their timers reduced by the amount currently on the turn taker: for example, a queue with living objects A, B, C and D with ‘next turn’ counts at 500, 950, 960, 970, would have A take its turn, lower the entire queue’s counts by 500 resulting in 0, 450, 460 and 470, then reset A to its default speed of 1000 and enter the queue again, ending up at the back of the line. Cycling through the whole queue to update this way each time seemed inefficient, and eventually led to edge-case errors involving ties and unintended negative ’next turn’ counts that were difficult to debug.

The system was eventually overhauled to adding a living’s object speed to their tick counter instead of resetting it each turn, leading to gradual increase of their update counter over time, as a full cycle of the queue would increase everything’s counter by 1000. This was an acceptable compromise, simplifying debugging greatly at the cost of limiting the turn count to about 2 million when a standard game usually lasts 5000 full turn cycles or so resulting in no change from the player’s point of view.

3.2.3. The Living class and being ‘alive’

Any object that has the potential to be an influencing factor in the game is a child of the Living class, named such for their potential to be living things in the game world. All Living classes are able to join the update queue to take turns, but those that aren’t expected to do anything on their turns such as walls can be designated as ‘un-alive’ at initialization to remove them from the queue. Requiring everything to be part of the Living class instead of making it an optional parent allows for more flexibility when converting things from ‘alive’ to ‘un-alive’ at will, such as if a wall was mutated by an earth-shaper to become sentient and defend itself from attackers, or if there was need for a regular unit to behave like a statue while retaining its other properties.

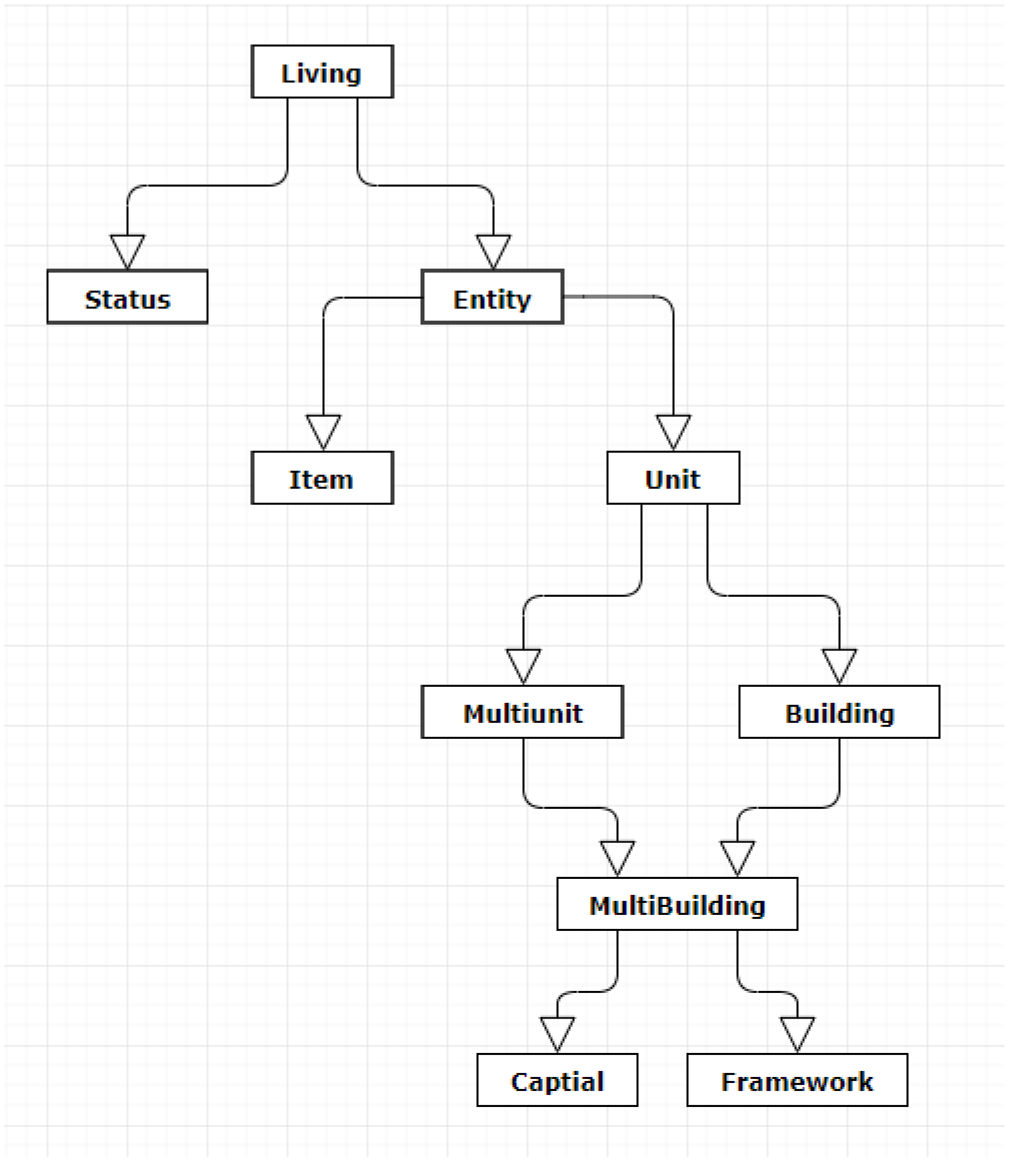


Figure 4. A summary of the Living class and its children.

Of all the Living subclasses, status effects are the only ones without a physical presence in the game world: they only exist as an attachment to units, still taking turns in the same manner but unable to be interacted with directly. Everything else that has the potential to take up ‘space’ on the grid is part of the Entity subclass, with X/Y position attributes to represent their location. From there, items are given their own class: they can be picked up by units and interacted with in common ways such as being equipped, used or thrown.

Units are the most common class, having a variety of ways to interact with other units and items in the world. They contain a list of tasks and memories used to determine how they behave. Units may also be part of the Building subclass, having limited movement but the potential to be constructed instead of just spawned in, or the Multi-unit subclass for things that occupy more than one grid-tile at a time. Buildings which occupy more than one tile are assigned to the Multi-building class. The Capital class is for the main HQ of a team, containing multiple helper-functions for dealing with allies who interact with them. Framework is a single class for all buildings under construction; when the supplies are delivered to the framework and the workers do enough build actions, the framework is replaced by whatever building was intended.

3.3. Units

Units are the most common class type, and despite the name can represent a person, inanimate object or any sort of non-humanoid creature that can exhibits behavior in the game world. Units employ a variety of attributes to determine their form and function mentioned in the following pages, but are the primary focus of many other mechanics of the game described in later sections.

A complete list of implemented units is provided in Appendix H.

3.3.1. Heath and combat stats

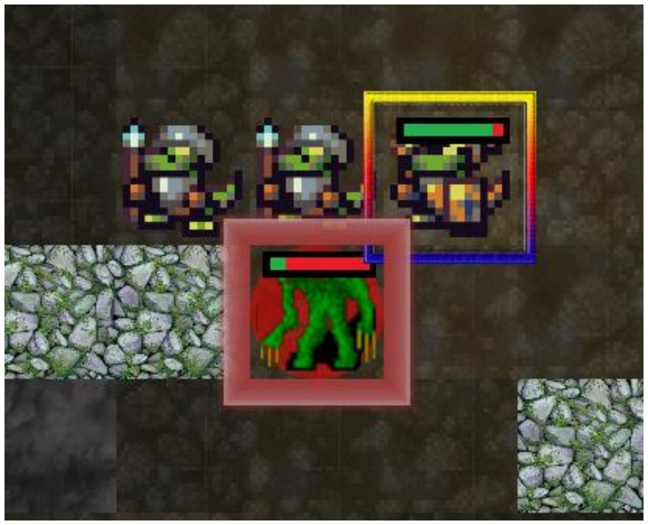


Figure 5. Units at less than their maximum health show their health bar, with the proportion of red to green indicating how much health they're missing. Source: Screen capture.

Health, or hp, is certainly important: when a Unit's hp drops to 0 and they don't have any special abilities that can save its life, they're automatically removed from the game. Each unit remembers its maximum health as maxHp, to determine how injured it is and the cap on how much it can be healed before overflowing. Figure 5 shows the bars used to indicate a Unit's hp status.

For combat, damage is divided into four types, Physical, Magical, Poison and Pure, together with their opposite defensive stats, Defense, Negation, Resistance and Divinity, that determine how much hp is lost when units attack each other. Each attack type is reduced by its opposite type <u>and</u> its current Divinity, to a minimum of 0 each, then added together and subtracted from hp. For some units, their default attack and defense values will be very low, and depend on their equipped weapons and armor to replace their weak stats.

3.3.2. Carried items

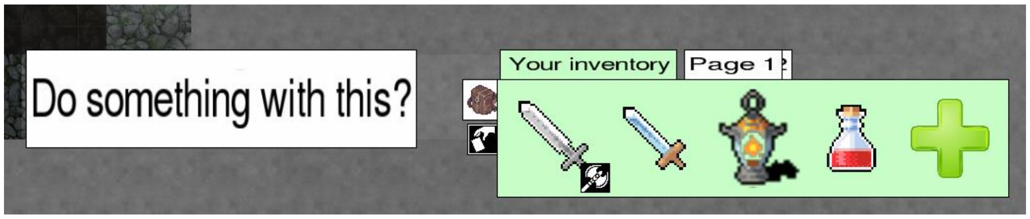


Figure 6. The player's inventory screen. If holding more than 5 items, an option to scroll to the next page is indicated by the green plus sign in the 5th position. Source: Screen capture.

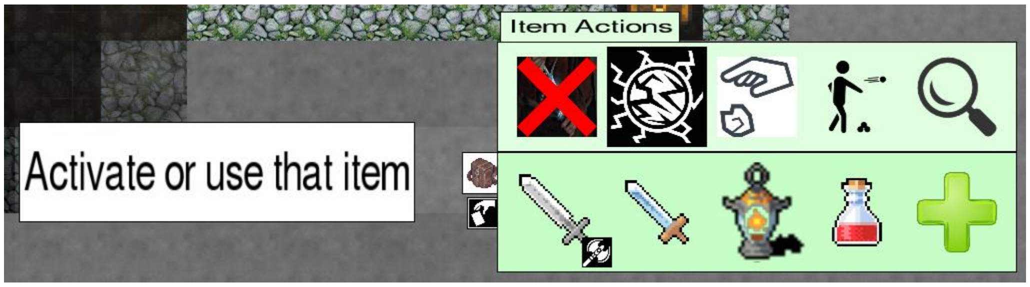


Figure 7. Clicking on an item in one's inventory brings up all possible options one can do with the item. Some options may be unavailable, like trying to equip a weapon you don't have training to use, or using an item that has no purpose. Source: Screen capture.

Units can carry items, the exact amount varying from unit to unit. These items are considered as part of the Unit, following their movements and accessible at any time. A unit can designate a single weapon or armor among the items they're holding, replacing their default attack or defense with the new weapon / armor's values, but require expertise about that type of item to be able to do so. For example, a typical priest won't be able to equip a heavy steel shield or use a bow, but are able to wield and use magic staffs that most others cannot thanks to their mystical training. Figures 6 and 7 illustrate how the player's inventory is displayed.

Units used to have a designated slot on their person for weapons and armor, making them not count towards the amount they were carrying, but was changed to the above version of simply keeping track of which ones in their inventory were equipped. This made it easier to code searching through items on a unit which helped the debugging process greatly, and was somewhat more intuitive for making the total held items count include arms and armor.

In addition to their list of held items, units also have a list of organs that are held the same way as regular items, but can't be used or interacted with in the usual ways. Organs typically only represent what they're going to drop on the ground when they die, such as an OreWall dropping its 'organs' of ores and gems once mined.

3.3.3. Basic interactions

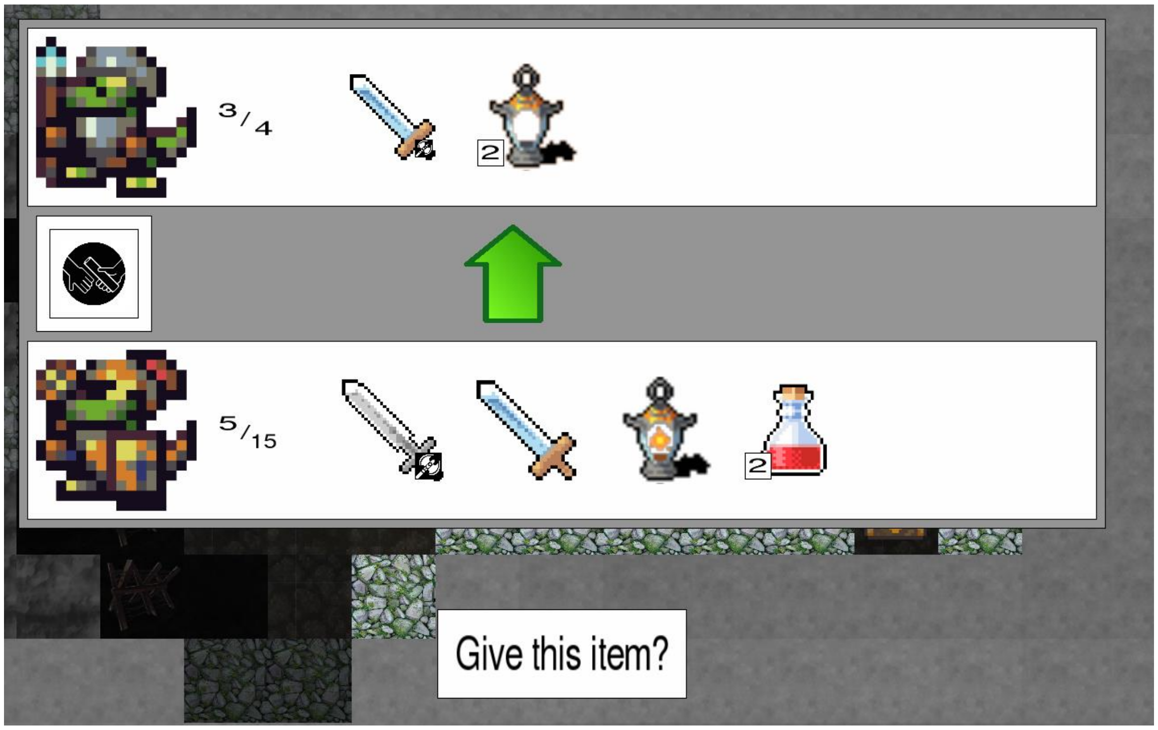


Figure 8. The trading menu, allowing the player to give or take items from allied units. Each unit's maximum carry amount is on the left, and going over that number and closing the window will drop extra items on the ground. Clicking the button in the left-middle changes the mode from giving to dropping, in case a player just wants their ally to drop their inventory. Source: Screen capture.

Units have a few ways of interacting with objects around them or on their person: picking them up is a start. If a unit is on the same square as an item they can pick it up, moving it from the ground to their list of held items, which hides it from the rest of the world for anyone doing a common search for items on the ground around them. Dropping works the same way in reverse. Figure 8 shows the interface used for item trading between the player and other units.

Units can try to equip items, with the same limitations mentioned before, or attempt to use them if they have any activatable abilities, such as ‘using’ a held potion to drink it. If other units are adjacent, one can try giving their items to another to transfer ownership and location.

Items can be thrown towards a location or other Unit; a raycast check is made in the target direction, and if nothing is in the way the item is removed from the inventory and lands on the ground at that spot. If a unit is hit instead, whether manually targeted or accidentally hit along the way, the item deals its specified thrown-attack damage to them and lands on the first tile between the victim and the thrower.

3.3.4. Teams and threat levels

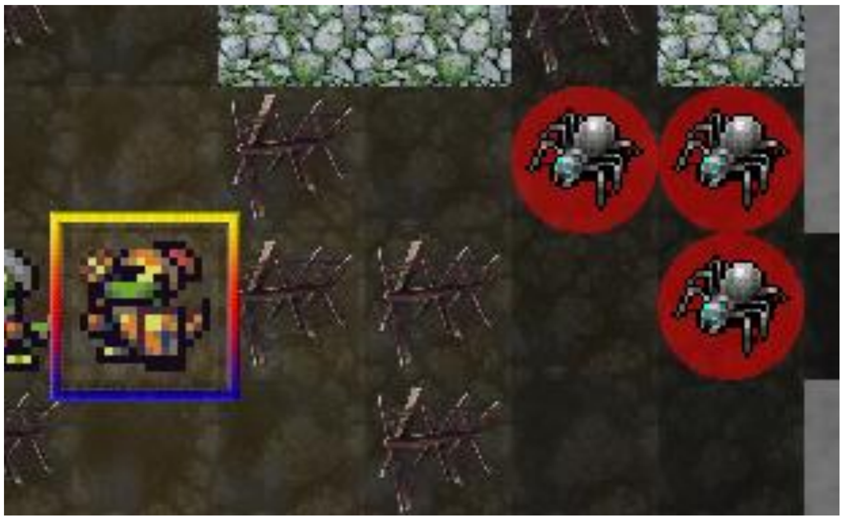


Figure 9. Enemy units come with a red circle to indicate hostility. Ideally allied units should also come with an indicator, but seemed unnecessary for the tutorial when allies were clearly the only other humanoids. Source: Screen capture.

Units always have a Team, even if they’re not in one. Used for determining who is an ally or an enemy, the current teams are Goblins, Humans, FeralSpiders, Spiders, Neutrals, Creeps and FeralCreeps, each inhabited by usually one type of race or overall theme of units. The exceptions are Neutrals and FeralCreeps. Every team is neutral with Neutrals, such as walls and bats, and won’t see them as an enemy to be feared (though they may attack them for other reasons), while FeralCreeps consider all other units as enemies, including other

FeralCreeps. Figure 9 illustrates an example of how enemy units (in this case, spiderlings) appear onscreen.

In addition, units have a Threat level that represents how dangerous they are to their enemies: 0 is a pacifist, 1 is completely subdued, 2 is temporarily subdued, 3 is a low-risk danger, 4 is an active threat, 5 is a high-risk threat and 6 or more is something unspeakably horrifying. The amount of bravery or cowardice towards an enemy is usually aligned with their threat level. Fighters prefer to fight active level 4-5 threats before dealing with helpless 3-threat farmers. Those weak farmers would behave normally near a hostile dragon if it were knocked out and locked cage, reducing its threat to 1, and only the most well-trained soldiers won't run in fear from a scary demon with threat level 6.

3.4. Items

Items are entities like units, existing as a physical presence in the game world and taking up space, but are smaller and more flexible about how they're used or moved around. A complete list of implemented items is provided in Appendix I.

Items have an attack and defense value, even though they can't be targeted by attacks or directly attack anyone by themselves. As discussed in the unit section, those values are meant to replace the owner's for as long as the item remains equipped. They don't have health and can't be destroyed in the same way units can, only being destroyable with certain interactions such as food being eaten or crafting materials being used to make the finished goods. As entities they exist on a square in the game world, but potentially infinite can be stored in a single square at once and don't block most units from moving over them.

Units have an additional property bound to their person and determined on a class by class basis: whether items in their inventory are being held or stored. Items are aware of whether they're being held, stored or an organ of whoever owns them at the time, and may modify or deactivate their normal behavior if they're not being carried in the intended way. This was done to ensure units made for carrying large amounts of items like carts or buildings don't get unfair advantages from being able to hold so many relative to other units. For example, the telescope item passively increases the holder's sight radius if held, but if dozens are placed in a stronghold for safekeeping they won't increase its sight radius to cover the whole map due to it 'storing' items instead of 'holding' them.

As items are a child of the living class, they can part of the update cycle and take turns like units. The vast majority don't, instead being static items that simply exist to be used by units, but exceptions exist such as meat degrading to rotten meat if they aren't being stored away in a building. When items take their turn, they only do whatever's in their class's personal hardcoded update function, as opposed to how units work with their task-oriented system (see the Tasks section).

3.5. Status effects

Status effects are intangible conditions that are attached to a Unit, affecting them without actually existing in the game world. While without any real form, other Living objects can still recognize status effects on other units or themselves and possibly react to them, or even attempt to prevent them from occurring in the first place. Status effects have a duration that indicates how long they'll last before expiring, though the way they count down is variable, and in some cases are permanent instead.

Status effects used to be attached to items as well in the same way, but the small number of necessary use cases and difficulty in keeping track of which item was which in debugging moved them to be Unit-only.

A complete list of implemented status effects is provided in Appendix J.

3.5.1. Duration

The duration period of a status effect typically starts at some predetermined number of turns, but can tick down in two different ways: having their own internal timer which adds them to the normal turn-taking cycle like normal, or becoming a static status that instead waits for its unit victim to take its turn before acting and counting down along with it. These different methods are used on a case-by-case basis: a magical fire lasting 5 turns should update independently and be Living, as one expects a fire to burn at the same rate on a slow turtle or a fast bat, and expire at the same time if cast on both at once. Meanwhile, for a confusion spell that makes the victim move in the opposite direction they intended, it may be better to make everyone affected always perform x steps this way, regardless of how fast or slow they are, thus a turn-by-turn timer should make it un-Living. This is primarily a concern for what makes sense from a player's point of view, though in most cases the descriptions of what's happening with each status effect should be intuitive enough.

3.5.2. Status types

Status effects can be different types depending on whether they're good or bad, temporary or permanent, magical or physical in nature, etc. In most cases temporary statuses are called Buffs if they're a boon or Debuffs if they negatively affect the victim, while permanent statuses are likewise called Traits or Curses. Status effect is a very general term, as the effects don't have to be mystical in nature: a peasant who's gone through military training can get a bonus to health and weapon skills with the Well-Trained trait, while a fighter yelled at by a scary ogre may have the Fear Debuff.

3.5.3. Status types

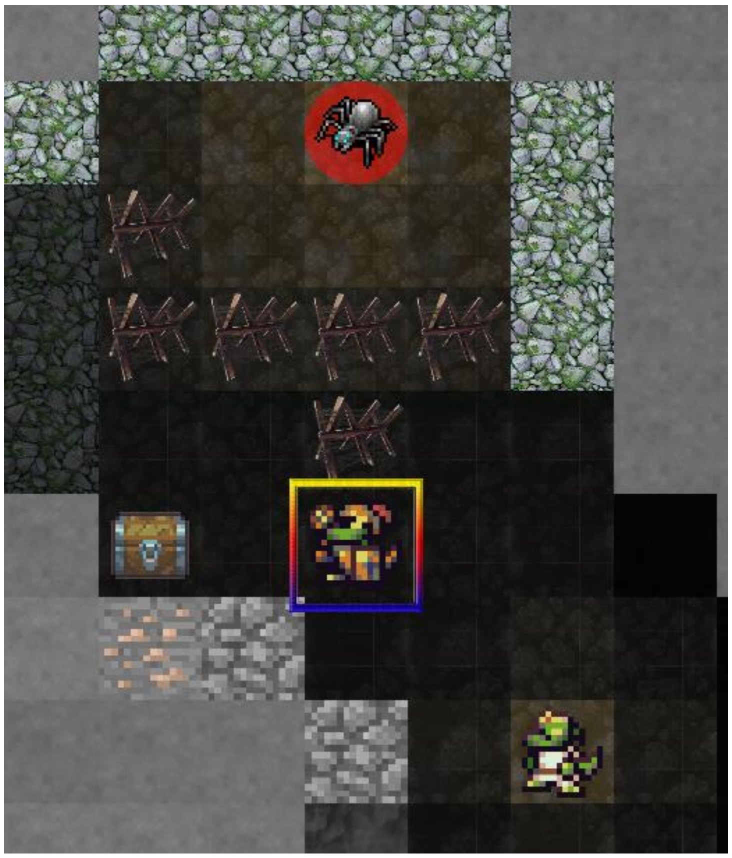


Figure 10. A priest ignites a contained spider with holyfire, a damage-over-time effect that removes invisibility from the afflicted unit and causes them to glow. Casting another holyfire on it would only increase the duration of the current fire instead of making a new one. Source: Screen capture.

Statuses can be prevented from infecting a unit before they happen, fully canceling out any effects they'd normally cause. The blocker will usually check for certain status types before rejecting their attempt to spawn on the victim, such as a priest's Ward status actively blocking a spider from injecting the Numbing debuff into a miner with its venomous bite. In addition, some status types may attempt to 'stack' their duration instead of creating another instance: a squad of 5 priests all casting the "holyfire" debuff on a single spider will result in a single, very long duration holyfire instead of multiple small holyfires (see Figure 10). In cases like these, the status will block any statuses of the same exact type on the same Unit, but add the intended duration to their own.

3.6. Tasks

Tasks are the primary way of making units do actions, and are highly flexible in terms of their priority or who / what they're attached to and when they're active / possible. A task is basically an action a unit can do on their turn, but are based on a logical behavior: a Run-From-Enemies task checks for nearby enemies and makes the unit flee if any are present, while a Drink-Potion-If-Low task will check that the unit has low health and is holding a health potion before attempting to drink it. Task queues were made to replace behavior trees when it became apparent that no amount of hardcoded behaviors were capable of keeping up with all the possible interactions being put in efficiently, and that the task-based system was much better suited towards making units more dynamically adaptable to their environment. A task queue also simplifies the decision-making process for players or debuggers: the order a unit will make decisions in at any moment is very clear, as well as where the decisions came from and whether they were relevant / necessary after having been done.

3.6.1. Organization

The important thing to note about the task queue is that it's a *priority* task queue: all tasks have a priority value ranging from 0-99999 that defines which are attempted first, initialized all at once in the World class at startup for easy comparison with one another. Those priority values, however, can be changed: Status effects, player decrees, or even the task itself can adjust its priority to be higher or lower on the fly, deactivated altogether or put on a cooldown for some number of turns / ticks. There aren't 99999 different tasks of course, but having such a large amount allows multiple to be put in a similar level of importance, thus allowing more flexible use for adding or subtracting priority value.

Priorities are by default separated into categories based on their value which usually signify their importance. Starting within certain categories assumes the task will adhere to certain standards, not explicitly checked for in the task-cycling function but followed as a general rule in creation. A unit taking its turn will attempt these tasks in order starting from lowest to highest: If a task fails for whatever reason the next one is attempted, and when one succeeds the process is stopped and the unit's turn ends.

- 0-9999 are for debugging tasks (for testing various things) and 10000-19999 are for 'ONLY' actions. Any task in these priority levels was made to be the unit's only possible action that it can attempt, and no matter what should end the unit's turn even if the task was unsuccessful. No 20000+ level task should ever be allowed to supersede these tasks, requiring thorough logic checks to make sure rising above that value is impossible, and that these tasks can't fall below it.
- 20000-29999 are for 'always' tasks, being lower priority than 'ONLY' but having about the same requirements otherwise. An example would be someone magically compelled to run in terror: if the victim's feet are rooted to the ground by some status effect, they shouldn't stop panicking and do other actions like normal just because they can't move.
- 30000-39999 are for dire needs. These tasks usually revolve around a temporary but imminent distress / impulse requiring immediate reaction. Tasks of these levels and higher are allowed to be modified by effects, and its fine if the task fails before others are attempted.
- 40000-49999 are for orders given by high-importance units, namely the player. Commands should be followed above normal behaviors, but in most cases non-combat-related commands will check to see if there's a battle going on and temporarily deactivate to let normal combat routines through.
- 50000-59999 are for combat-related actions. These tasks are used when a unit sees an enemy, and can include running away just as much as actual fighting as long as its what someone does in combat situations.
- 60000-69999 are for minor emergencies, such as emptying one's inventory if they're full before going out mining again. These are things a unit should deal with before continuing their regular duties, not necessarily being a bad thing.
- 70000-79999 are for normal duties, being whatever a type of unit is expected to do normally such as miners mining or scouts exploring. The result of their efforts usually

cause a 60000-level task to be activated when they’re done, allowing them to reset work again.

- 80000-89999 are for ‘weak’ duties, mostly meaning trying to find ways to get more of their normal work. Checking up at the capital is usually the thing to do in those situations, or any other popular information hub, or with whoever’s nearby as a last resort.
- 90000-99999 are for idle actions that one does if they have nothing else they can attempt right now. Mostly this is just wandering around in circles or exploring aimlessly.

Tasks of the same type tend to have similar priorities, but are further distinguished by tags that can separate them into factions like ‘cowardly,’ ‘violent’ or ‘greedy’ and such. Effects like a cowardice spell can be made to find all ‘cowardly’ tasks and increase their priority, making them attempted before regular fighting actions for example, or vice-versa with a ‘bravery’ spell. A player will naturally learn the order of actions a general type of unit performs, but should be able to do so intuitively even if the exact numbers aren’t available. Similarly, effects that change priorities like cowardice should be important, having both noticeable effects but also possible countermeasures or retaliatory actions to regain control of the priority system to their favor.

3.6.2. Cooldowns and counters

Tasks may seem like they should be connected to the energy-based turn system, but are in fact not part of the living class. Unlike items or status effects, tasks will never need to operate at a fixed time independent of their unit controller: they’re just a list of possible actions a unit can make, rather than something that exists in any form to take an action by themselves. For tasks on cooldown, they’re simply set to not be active until the tick counter stored in the World class is past a certain point; there is no need for them to enter the turn cycle to turn themselves on at the exact right time, as they can always just wait to be checked up on later when actually being called.

If a task should be set to only activate after a unit has had x-many turns, the solution is also simple enough: add a value to the class every time it tries to call that task, decrementing till 0 after x many turns before allowing itself to activate again, ensuring the cooldown is tied with the Unit's update loop. The unit also saves the last task it attempted for similar reasons: charging up a spell over multiple turns can check to see if it was used last time, incrementing an internal counter for however long is necessary before activating, or resetting to 0 if something else took priority or they were otherwise interrupted.

3.6.3. Items and tasks

Items have a list of tasks, though they don't perform the tasks themselves. When items take their turn, they only do whatever's in their personal hardcoded update function, with most items being non-updating static objects like rocks or meat, though exceptions are possible (such as a bomb with a lit fuse). Instead, these tasks are added to whoever picks up the item, which the owner attempts to carry out with the rest of their usual tasks when they update. This adaptive behavior allows units to use carried items intelligently, even if they by default have nothing to do with them. For example, a miner by default doesn't go and punch rocks, but only knows to go out and find a pick. When one is found, the miner is given the pick's list of tasks that involve mining ores and bringing them back to a nearby storage area. This allows units to be general purpose without wasting time failing tasks due to supplies they don't have, and items to always be used by any appropriate unit that has them. These tasks sometimes depend on being held a certain way as mentioned in the items section, or that the unit be of a certain type. For example, if a pick is given to a cart that stores multiple items at once for transport, it won't receive the pick's task to equip it due to being 'stored' instead of 'held'.

3.6.4. Status effects and tasks

Status effects have a list of tasks added to a unit in the same manner as an item, remaining for long as the status exists attached to that unit. Most tasks added by a status effect have high priority levels around 20000-40000 that take over a unit’s normal actions for as long as the status is there, but more permanent effects like traits may add normal low-priority tasks a unit will attempt along with their normal behavior.

3.6.5. Makers and doers

Tasks can be added to units via items or status effects, but are bound to that item / effect and usually only last for as long as the bound thing remains with the Unit. Tasks keep track of both the unit they’re currently attached to and the thing that created the task in the first place. This helps a player understand what added certain behaviors to units if they start acting atypically, and allows task creators to pinpoint their own creations. This also allows the creator to keep track of things related to its task: a sword that gains power as it’s owner kills using it can increment its kill count every time it’s attack task is used to good effect, modifying the task and remaining that way even if the task is removed and implanted into the next unit that picks up the sword.

3.6.6. Commands

Player-created tasks are referred to as commands. No matter what type of unit a player is acting as at the time, they’re able to command nearby allies by giving them one of numerous preset tasks that usually overrides most normal duties on their priority scale depending on the command. The tasks are general purpose, including following a selected unit, moving towards a location, picking up all items around a given spot, and many more, each with levels of caution concerning dangers they find along the way ranging from suicidal tunnel-vision to immediate retreat.

Commands used to be queue-able, but very few use cases and an extra-click worth of complexity on the players part made it seem unnecessary. Instead, new commands overwrite previous commands given by the player. When the commands are fulfilled, they simply delete themselves from the Unit’s queue with no immediate feedback to the player; the results have to be discovered indirectly through the unit reporting their memories of the results or the player discovering what happened.

3.6.7. General purpose

Tasks used to be a comprehensive set of everything affiliated with a behavior; a mining task used to include looking for ores, looking for walls, mining walls, bringing the ores back to base and checking for updates about where new walls are all in one go, but it soon became apparent these were better split into their own separate actions that came with whatever caused a miner to act like a miner. This started making tasks more like ‘actions’, which allowed them to be used for things like casting spells or using special moves as well as mimicking overall behaviors.

3.7. Attacks

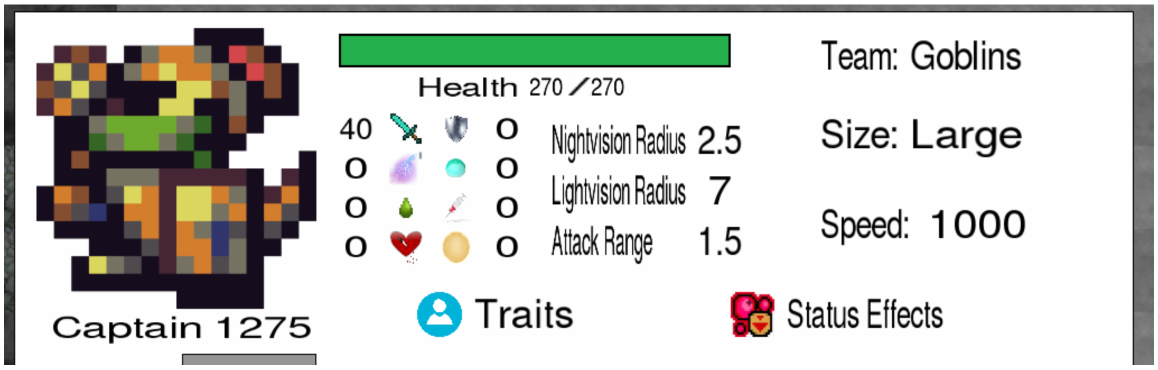


Figure 11. The statistics page from the description menu shows most possible stats associated with a unit. The attack and defense values, next to and below the sword and shield icons, are based on the weapon or armor equipped instead of the unit’s default values. Source: Screen capture.

An ‘Attack’ not only have separate attributes like physical, magic, poison and pure damage, but contain their own class as well, encapsulating everything about a single complete ‘attack’. The attacker, target, distance, weapon used, attack type and attack result are stored in this class, for the sole purpose of allowing other things to react to it. Figure 11 shows the onscreen display of attack-oriented statistics associated with a Unit.

When a unit takes a swing at another unit, a copy of their current attack value (either their default one or their equipped weapon's) is created, taking the effects of any outside modifiers that may increase or decrease the four standard damage attributes. Those attributes are then reduced by the target's defensive stats, and the sum of the remaining damage is removed from the target's hp. This process allows the attack to quickly be shared globally with things that may react to an attack being launched without having to modify the original attack values back to normal each time.

3.7.1. Engaging and targeting

Attacks require the target to be in range of the attacker, which doesn't always mean melee distance. For this, units have their own personal methods of engagement, allowing them to head towards the nearest enemy and attempt an attack regardless of their weapon or range. The engage function has inputs to intelligently use whatever weapon is currently equipped, to use only default punches for the good old-fashioned barbarian rage, and / or to target only a specific few units such as a miner looking to mine nearby walls but not enemy bats.

In the case of ranged attacks, a raycast attempt is made from the attacker to the defender before the damage process begins. Being able to aim at something doesn't necessarily follow up with a hit: it could be invisible units, their aim being redirected elsewhere, or the attacker just shooting blindly, but if something gets in the way of the destination they switch to being the target instead. In the case of multiple interruptions, only the one closest to the attacker is struck.

3.8. Squares

The grid of the game world contains an X by Y number of Square class objects, each used to hold data about who or what is contained within them. The grid is just a 2D array: there are no sub-squares or non-integer values.

3.8.1. Watchers and glowers

Squares are primarily a container for all the different types of things that can occupy them, which are units, items, ‘watchers’ and ‘glowers’. Units and items enter and leave the square only when they move in or out of it, but the other two are different: ‘watchers’ indicate which units currently have line of sight to the square, making then possibly able to see the item or unit inside, and ‘glowers’ indicate how many light sources are close enough to be considered in-range, thus making the square considered bright.

Whenever a unit enters or leaves the square, a check is done on all watchers: if the square is bright and within its light radius, or is within their dark radius, they notice the unit’s movement and add or update them in their memories. In the case of a unit walking out of a square, they also recognize where they just moved into and remember them as last being at the new square, even if they’re not ‘watching’ the arrival square. When a watcher or glower moves, all their current watched and glowed squares are removed, with a raycast check within their radius to check for their new updated positions. Note that if a unit’s light radius is farther than their dark radius, their watched area may extend far into the dark where they can’t currently see, waiting for something that glows to come along and make it visible.

3.8.2. Movement blocking and sizes

Squares are often checked to see if a unit is allowed to move into them, usually depending on how big the people already there are (see Figures 12 and 13 below). Unit and items are entities, thus occupying one or more squares in the grid, but additionally have a size property that determines how many other things of other sizes should be allowed to share the same square, with a few exceptions. Units have a size value determining how big they are, while items will always have a default size: 0 for ghost-lik, 1 for gaseous or flying things, 2 for items or very tiny units, 3 for dog-sized, 4 for human-sized, 5 for giant-sized, 6 for building-sized, 7 for solid walls and 8 for magically sealed walls.

The above size list isn't an exact measurement; its primary use is determining whether a unit can move into another square, considering the sizes and team affiliations of the units in the impending move.

- Size 0 units can go anywhere except a square with at least one 8-sized unit.
- Size 1 can go in squares that don't have at least a size 7.
- Size 2 can move where the largest size is less than 6.
- Size 3 can go into squares with only other size 3's or below, but are blocked if one of the size 3's in the square belongs to an enemy: they'll reject the mover, blocking and pushing them back.
- Size 4 and 5 can only move if the square's largest size is 2 or below.
- Anything with a size higher than 5 normally shouldn't be moving at all, but are otherwise blocked by size 1 units (except size 8, which is blocked if anything is in the incoming square at all).

Every movement attempt, each unit in the occupying square is checked against these values. While that might sound intensive at first glance, usually only one of these checks ends up being done, and squares rarely have more than one unit in them for checking against.

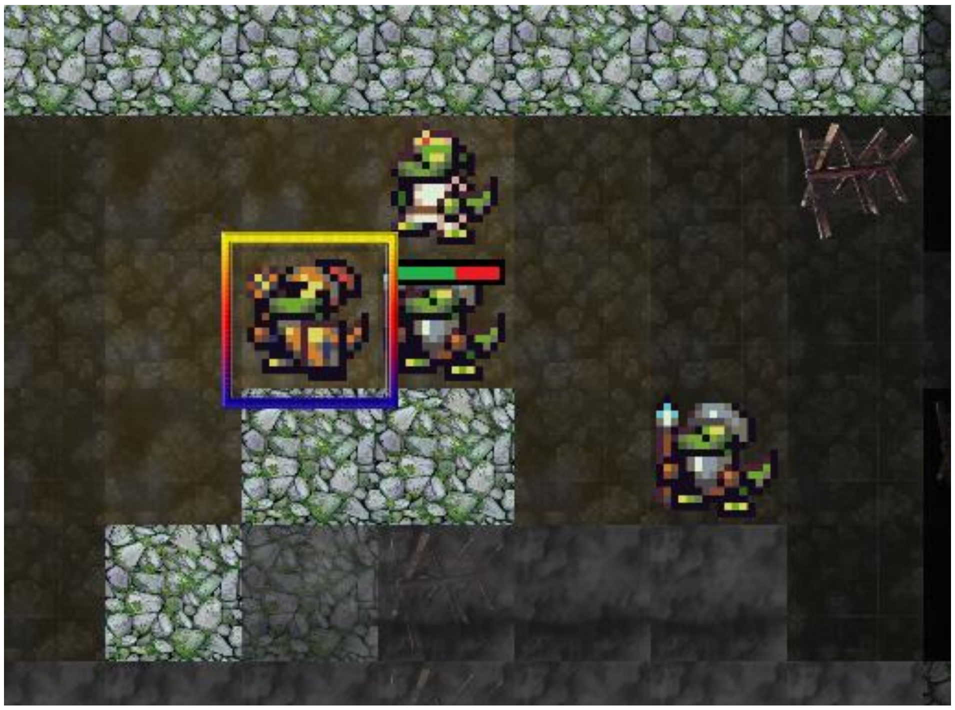


Figure 12. A player is followed by three medium-sized combat units, but is creating a bottleneck in a 2-square-long hallway. The third soldier follower can't get through. The player is able to manually push allies out of the way to path through them, but normal allies can't. Source: Screen capture.

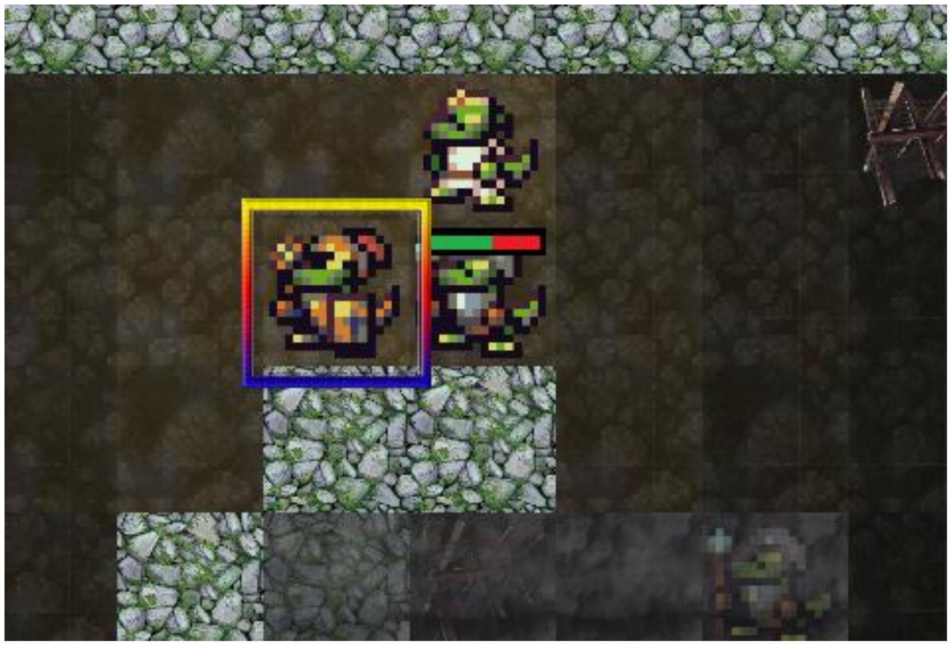


Figure 13. The soldier will instead run around and try to find another path to reach the player, usually ending up losing sight of him and then heading back to the capital to try and find him again. This process has caused a massive amount of confusion and grief with playtesters. Source: Screen capture.

3.8.3. Being ‘hidden’ from the grid

Entities have an additional property in addition to existing on the grid: not existing on the grid. Primarily used by items, the ‘isHidden’ value determines whether an entity is currently where it’s supposed to be in the world at its x/y coordinates or merely representative of where they should be, preventing interaction with it or searches directed towards its type.

When a unit picks up an item, the item become hidden and stops existing for other units to find and pick up themselves, but still exists as an item being held by its owner and can still be found via checking the owner’s inventory. This means any location-dependent abilities are still accurate wherever they’re carried, or anyone seeking out a hidden object with pre-defined knowledge about what it is can still have a goal to move towards.

3.9. The movement process

Movement, from either items or units, behave a lot like Attacks in the number of layers and reverse layers they go through to completion. All of that Unit’s currently watched units are cleared, its memories are set to a minimum of “I just lost sight of everyone,” and every square that contains it as a watcher is cleared. Squares that contain it as a glower are also cleared, but recorded for later use. The mover is then removed from their current square and placed at the new square, with all their items following suit immediately after. Glowing is then reactivated, checking for all the possible squares that can be raycasted to, and adding the mover as one.

If a square that had no glowers before the move gained one due to the movement, or a square just lost its last glower, they refresh the sight of all their watchers to check if they lost sight of or gained sight of anyone at the now-changed square. The mover then raycasts out their sight radius the same way, adding themselves as a watcher to the new squares and updating its memories to account for the new discoveries. The Unit’s movement is then made known to the watchers of its previous square and the current watchers of the new one, and then the World cycles through all Living objects that check for reactions based on movement to see if any care about the mover's new position (such as pressure-plate traps being sprung if a unit steps on them).

The above methodology is necessary for two very frequent cases: a sight-blocking unit moving with viewers on either side, and something with a very large glow radius causing the game to lag to a halt. The above technique of only checking for the updated glowing squares solved the latter problem, but for the first imagine a long, narrow, one-square-wide corridor: a high-sight-radius scout waits at the very top, a fat view-blocking fog-demon is just below them, and a bat lies at the other end of the corridor. The above process was the result of many attempt to make it so the demon, when moving down one tile at a time, would still be blocking the line of sight of the scout: for the longest time during movement, the scout would be able to glimpse the bat for a brief moment and record it in their memory when the demon 'disappeared' from his starting square for a single cycle to move to the next.

3.10. Pathfinding

Pathfinding has been a long process throughout the project's history, not due to any experimental new techniques, but because of constant attempts to find a way to account for every imaginable scenario of pathing from A to as close to B as possible while maintaining efficiency. Pathfinding started out completely breadth-search style, then was quickly renovated to A* and has remained so since. Because diagonal movement was counted the same as horizontal or vertical moves and the closely-bounded level designs, other pathing techniques tended to be inefficient or inaccurate when tested.

Common time-saving techniques are implemented, such as checking if one is near their destination already, or pre-defining navigable areas beforehand. There's a distinct difference between pathfinding to an exact square or just trying to path to any square around it, and for pathfinding to any number of target tiles as opposed to only accepting a single end point.

3.10.1. Planning a movement vs actual movement

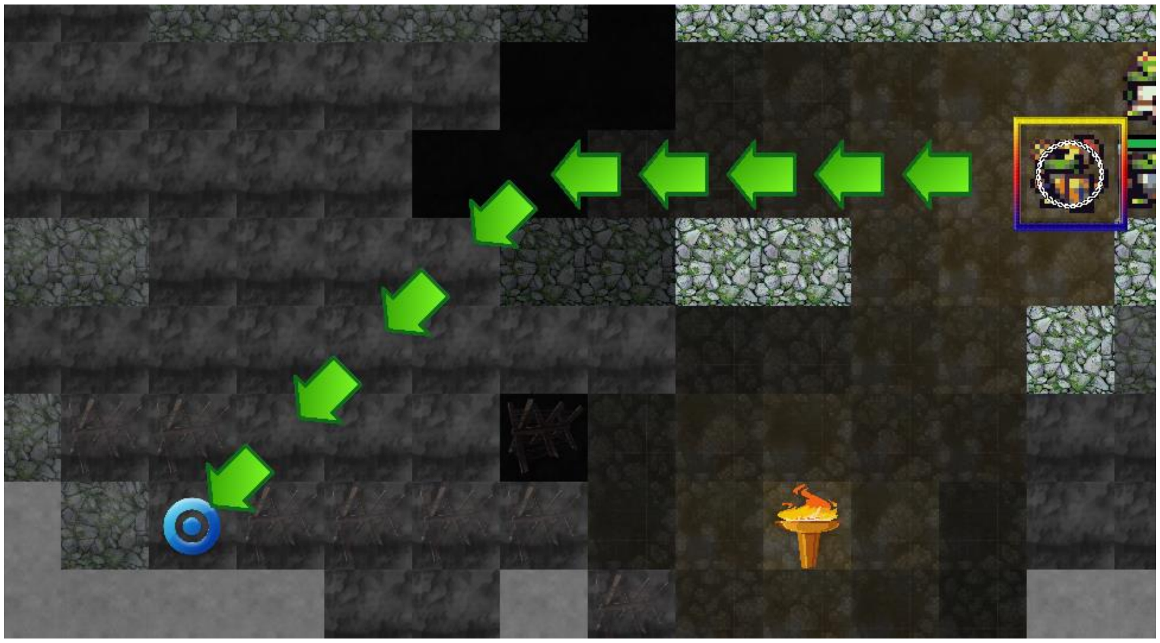


Figure 14. An illustrated path of where the player plans to walk towards, pathing around the rock walls in the way and between the two wooden barricades in the fog ahead even though he only has memories of them. Source: Screen capture.

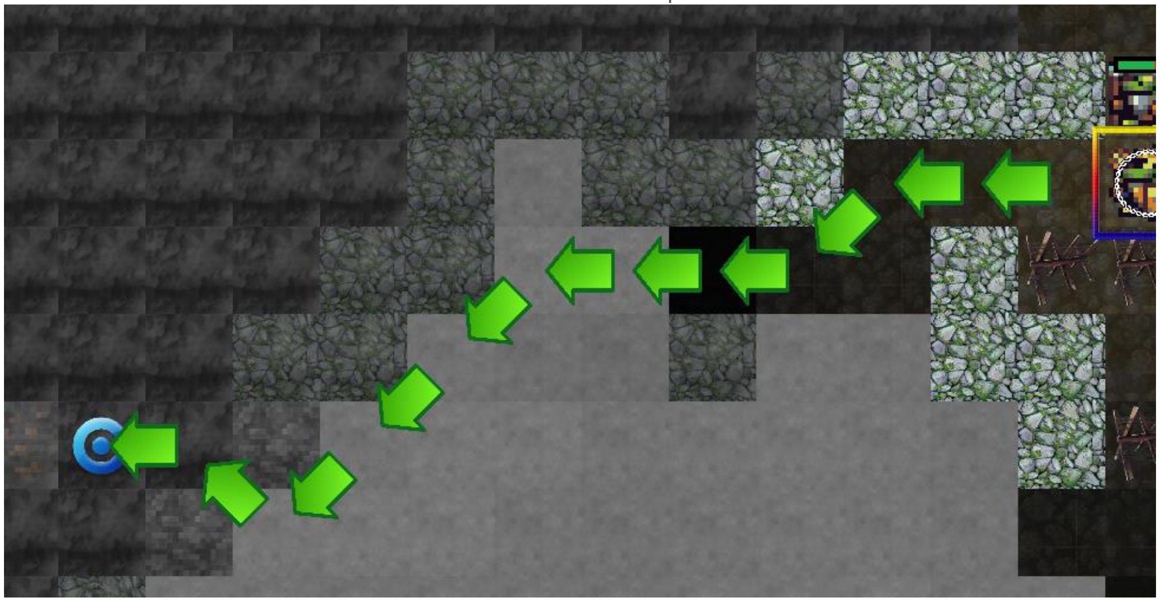


Figure 15. A problem that came up during playtesting is that the path wasn't always illustrated. While the shape of some structures and long corridors of walls are obvious to the player, the pathing algorithm assumes anything in the fog is passable if it hasn't been explored yet and frequently routes through areas that most likely are blocked off. When the wall is discovered, another path is immediately rerouted, possibly heading in an even worse direction. Source: Screen capture.

What about invisible units, or squares the unit has seen before but can't see right now? Planning to move into a square is different than attempting it, and the results may differ if one doesn't have perfect information about the destination. This allows units the mover doesn't notice at the moment but should block movement to go undetected, making the mover believe they can attempt the move and in doing so bumping into the blocker instead.

Actual attempts to walk can only happen into squares adjacent to the mover, but any square can be checked to see if the mover thinks they'd be allowed to move into it. When planning a path some distance away that goes beyond their sight radius, memories come into play. Rather than checking all the units currently at an out-of-sight square, the mover's memories are checking instead, looking for anything last seen at that location that was last known to be able to block the user's move. In the vast majority of cases this means a wall blocking the way, and the mover will remember to try and find other squares to path around rather than hope a solid wall wandered away while their back was turned, as illustrated in Figures 14 and 15.

3.10.2. The 'Path' result

The return value from a pathfinding function is a 'path' in the form of a queue of squares leading to the target destination: an empty queue means the unit is already there, a queue with a single square set to negative x/y values means a path couldn't be found, and a queue filled with normal squares shows the steps needed to walk from the pathfinder's current position to the closest destination. This means a typical attempt to see if someplace is navigable requires a manual double-check immediately afterwards to see if there's any path to follow at all, or if the path is real or a fake one with a negative square that would cause immediate errors if attempted to move into. Making a 'path' object that would be returned instead would be the standard approach, but wouldn't have any apparent improvements on anything beyond a one or two line shortening from the template already in place to make pathing checks, so this potentially dangerous method remained without incident so far.

3.11. Claimed targets

The search for things to be pathed to is often more complicated than the pathing process itself. Units keep a record of everything they know in their memories, but often times they only care about things they can see immediately in front of them, which are kept a list of watched units for easier iteration. For example, when a miner is trying to mine a wall, they'll always head for the closest one they can see before searching through every memory for an out-of-sight mineable wall, on average reducing computation time greatly and making their movements more predictable / intuitive. Often it turns out that many other nearby miners have the same idea, meaning they may pile on excessively towards the same destination if it's the closest one to all of them. This was causing chaos when it came to picking up ores dropped on the ground; whenever one broke off a wall and dropped nearby, every miner would stop mining to scramble towards it when only one person needed to do so. This led to checking if things could be 'claimed' by other visible units of the same type by being the closest ones to said destination. Seeing that a target is 'claimed' meant the claimant was the same unit type as the seeker, likely doing the same things with the same thought process, and should be the one to handle that target since they were closer to it. This fixed the ore problem immediately, and also found some creative uses in combat where targets were preferably spread out among the fighters.

An important note is that units only consider other claimants they can see. If two units are after the same quarry but can't see each other, they won't consider the thing claimed by anyone else and both head towards it like normal.

3.11.1. Filtering functions

Finding claimed units was one of many necessary filters. Given a number of known items, units or memories, a given scenario could require them to be narrowed down in a number of different ways: only things farther than x distance away, only things with a certain status effect on them, only things holding x many items, only things of a certain type, etc. Manually scanning for these terms or conditions proved repetitive and error-prone, so filtering functions were added to the World class that would skim through vectors and only keep the desired objects. The current filters include: species, family, genus, order, visibility, see-ability (if a unit can see something regardless of whether it is trying to be invisible or not), raycast-ability (includes whether one can directly see-to, aim towards or fully target towards their quarry), within / outside of a given radius, team, tags, holding or claimed. A Unit's typical internal check for what it has memories also comes with some filtering options that can be left blank to ignore, containing these kind of filters: within a given radius, raycast-able, species, tag, team level (only enemies, anything not an ally, anything not an enemy, or allies only), threat level or status level (only dead memories, dead or missing memories, visible or out-of-sight memories, or only visible guaranteed memories).

3.12. Visibility

All units are capable of seeing things around them, but the process to determine what's viewable or not is a multi-step system that requires constant updates whenever things start moving around. The three main steps to seeing any entity are being able to raycast to it, checking if square is close enough to the viewer's light or dark vision radius, and making sure the target isn't invisible to the viewer.

3.12.1. Light levels

Light levels are synonymous with glowing: rather than setting a square to bright or not via some constant, all entities have a 'glow level' value representing the radius around them that's lit up at all times. The radius follows the entity, moving along with it, and spreads outward whenever the environment is refreshed to check for things that may block line-of-sight such as walls or smoke. Squares the light source reaches puts the source in its list of 'glowers,' and for related visibility checks the square is considered bright if it has at least one glower regardless of its distance. Figures 16-18 illustrates a player manipulating a light source (a torch).

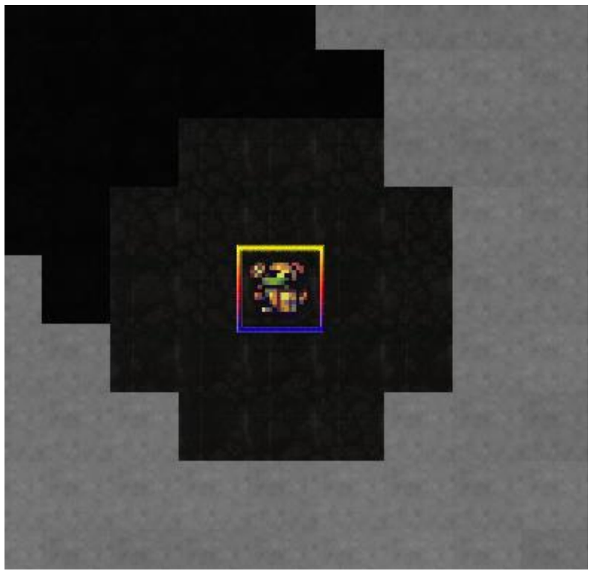


Figure 16. A player with no light source can barely see around him. Source: Screen capture.

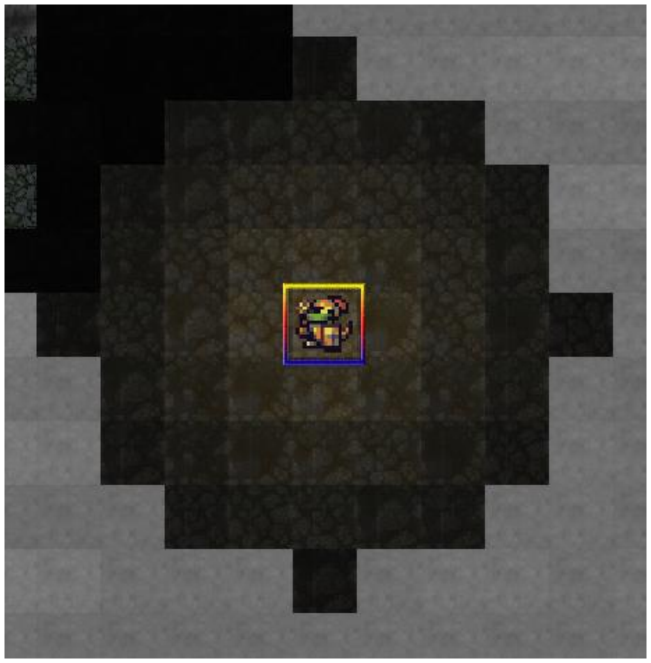


Figure 17. A player with a lit torch can see a greater distance. Source: Screen capture.

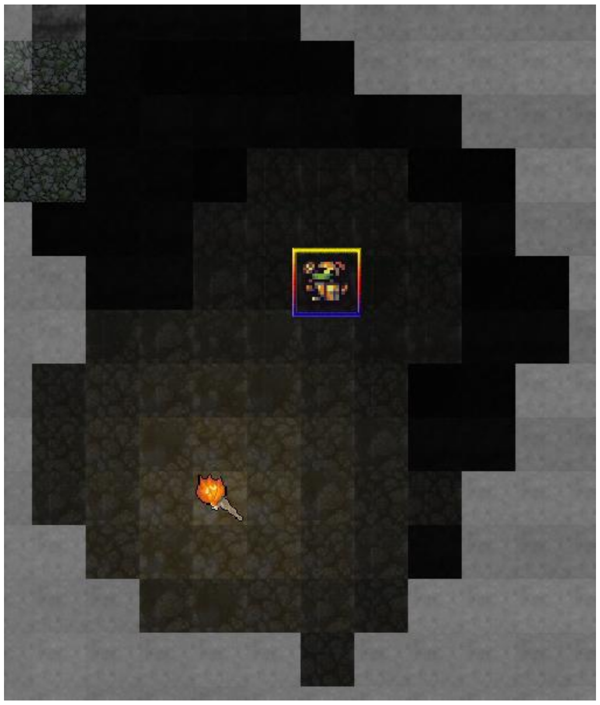


Figure 18. Throwing the torch makes the glow radius follow it, still providing its full circle of light. The reason it doesn't appear as circular as before is because the player's light radius vision doesn't extend far enough to see the outer edge. Source: Screen capture.

While all in-game logic only considers light sources as being on or off, when being rendered on a square grid this looks absolutely terrible (as discussed in the rendering section) and was made to only aesthetically consider distance from the source so that farther areas were dimmer than the square the light source was directly on. This gives the player the advantage of being able to judge where the source of a light is coming from, while AIs only recognize the lit squares and won't connect it to a circle-shaped source in the same way.

3.12.2. Sight / glow radii

All units can only see some distance around them through the ever-present fog of *Trick Of The Light*. The values that determine how far are their light radius and dark radius, indicating the range they can see in well-lit areas and in complete darkness respectively. Described in more detail in the Vision section (3.13), every entity has a Glowlevel that determines how many squares around them are considered bright, and a square without any glowers is considered pitch black. Figures 19-22 illustrate how light radii are displayed on the map.

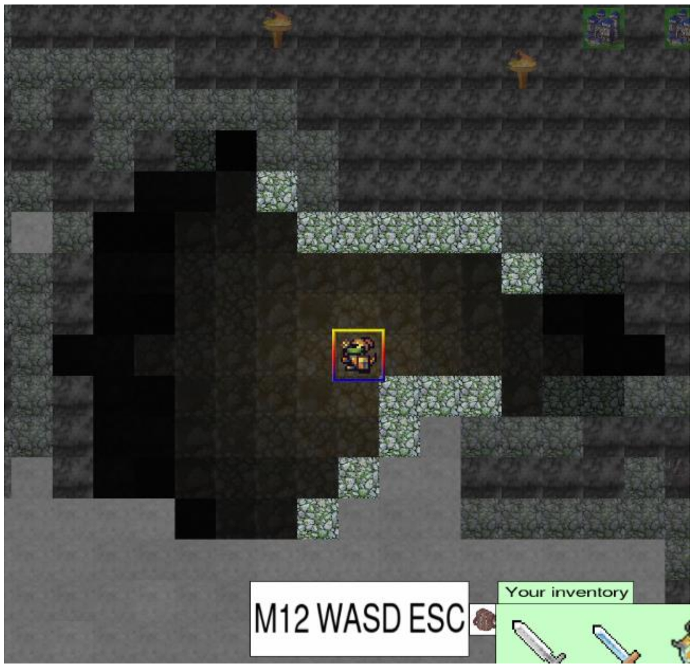


Figure 19. A player with a lit lantern can see a good distance around, but the light doesn't reach the edge of its viewable area. The pure-black but non-foggy edges of the circle represents an area that could be viewed if there was light there. Source: Screen capture.

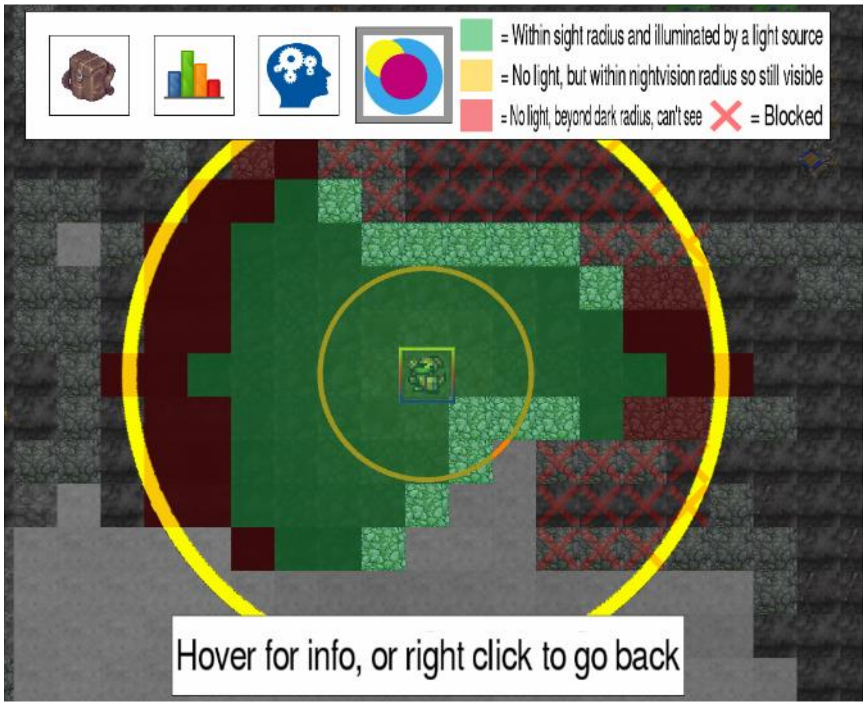

🟩 = Within sight radius and illuminated by a light source
🟨 = No light, but within nightvision radius so still visible
🟥 = No light, beyond dark radius, can't see ❌ = Blocked
Hover for info, or right click to go back

Figure 20. The vision-tracking focus lets players see the specifics about their sight radii, including why they can or can't see specific tiles around them. Source: Screen capture.

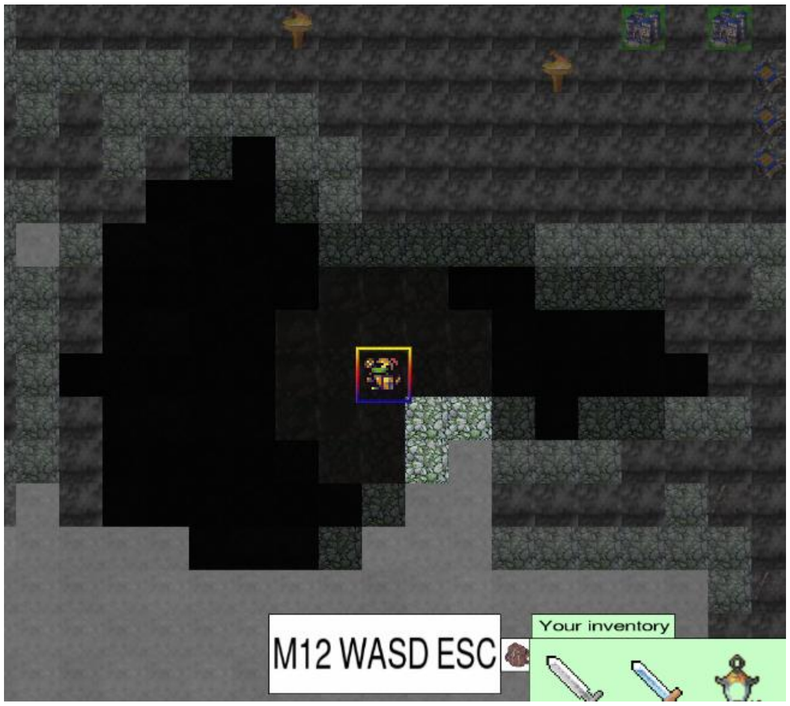


Figure 21. When the lantern is turned off, the player stops emitting light and can only see as far in the dark as their dark radius allows. Source: Screen capture.

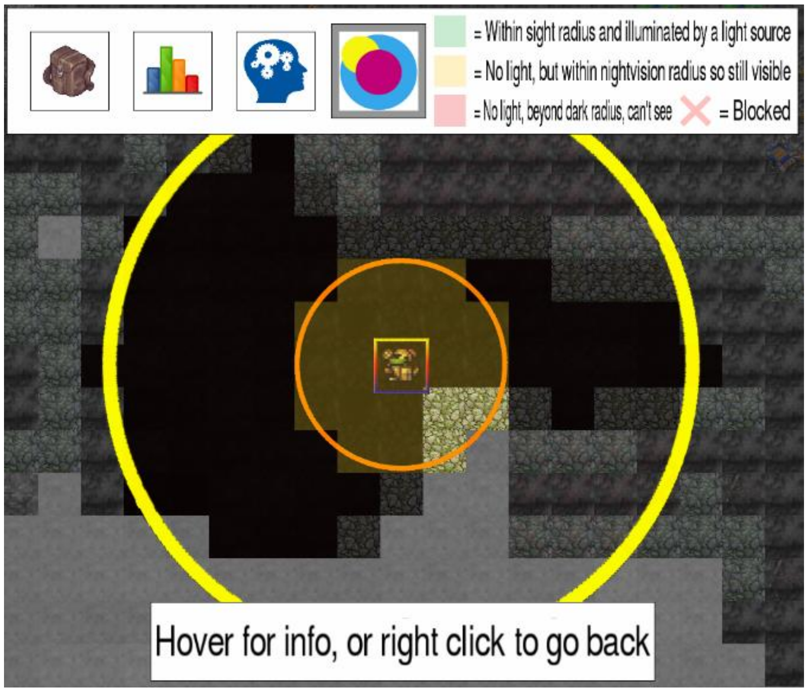

🟩 = Within sight radius and illuminated by a light source
🟨 = No light, but within nightvision radius so still visible
🟥 = No light, beyond dark radius, can't see ✕ = Blocked
Hover for info, or right click to go back

Figure 22. The player's orange dark radius is smaller than their yellow light radius, but that's not always the case. Some creatures see farther in the dark, making their light radius useless. Source: Screen capture.

If a square is bright and the distance between it and the unit is less than the light radius, or if the square is either bright or dark but within the dark radius distance, then that unit can see the square and tell what items and units are within it. This means the dark radius is strictly better than the light radius, but for most units the light radius is much larger, meaning a well-lit area will be much more visible than a dark one. These mechanics result in light sources being a double-edged sword: carrying something bright like a torch will help one scout a greater distance around them as they travel, but something in the darkness ahead is likely to see the torch-bearer with his light before being seen in the darkness.

3.12.3. Raycasting

To see whether someone is invisible or not, one needs to be able to see to where the intended target is. To know the viewable area, given a Unit's sight radius, all squares in range are raycasted to check for anything that would block line of sight.

All sorts of methods were tested for what worked best for raycasting, always only concerned with having the ideal output no matter the computational cost, which ended up making it the most expensive algorithm overall in terms of CPU use. Starting as a simple check copied almost directly from the Roguebasin-wiki tutorials, typical ray casting consisted of checking if the center of the starting square could draw a straight line to the receiving square without being interrupted. (Roguebasin, *Register*) This immediately brought to view a problem with long series of walls: gaps would appear after a certain distance as if you couldn't see half of the wall straight in front of you. Shadowcasting, again via tutorials from Roguebasin, were implemented instead, fixing the previous problem but causing new ones, such as being asymmetrical (standing in square A and seeing B didn't always mean one could see from B to A) and lone wall 'pillars' not covering things behind them in rational ways (usually with the immediate back of the pillar being visible but further squares being blocked). (Roguebasin, *Bergström*)

Diamond raycasting was tried next, but before a complete replacement was finished again it was apparent that the logic wouldn't work with the gameplay. Diamond raycasting treats pillars as being diamond-shaped for movement and vision, but would only be ideal with diamond-only movement and accounted for a larger gap between diagonal squares than desired. (*Milazzo*)

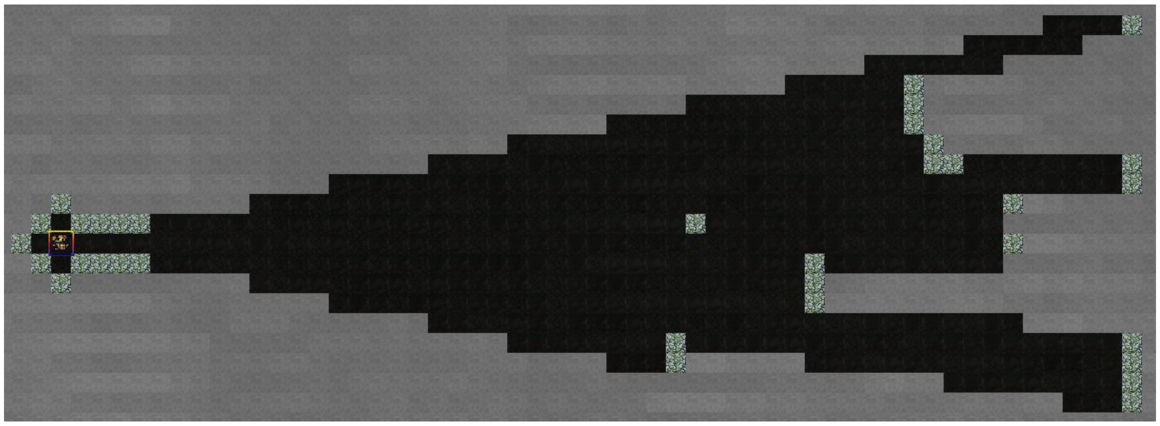


Figure 23. An example of a cone of vision extending outwards. Note the symmetry between upper and lower bounds, and how the walls near-adjacent to the seer don't reveal themselves unless the player is exactly diagonal with them. Source: Screen capture.

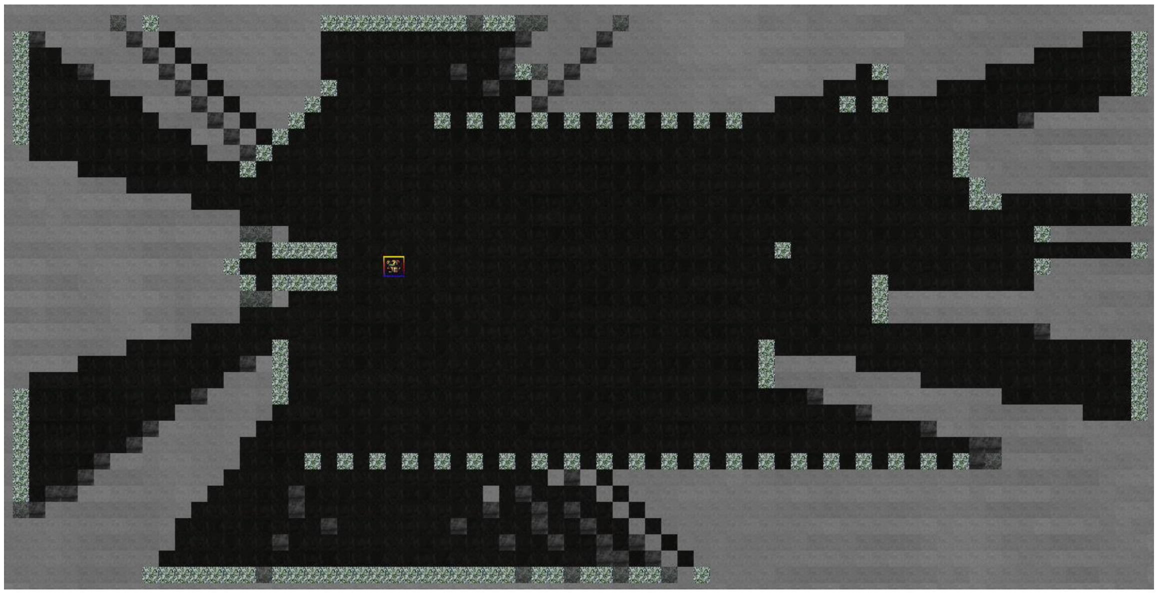


Figure 24. Another view further outside the narrow tunnel. Vision will never extend this far in a normal game, normally being around 3-9 squares maximum, but it is important to make sure vision works correctly at every distance. Source: Screen capture.

Eventually, all work was moved towards a heavily edited version of normal raycasting, drawing heavily from the *Dungeons and Dragons* version of sight and cover checking. Instead of just center to center, every octagonal point to octagonal point was checked as well with special checks for exact-diagonal-edge cases that worked with our movement system (see Figures 23 and 24). Halfway through testing it became apparent that using doubles were giving rounding errors for our approximate calculations by being unable to represent ratios accurately, causing eventual overflow or underflow that caused constant irregularities and made the algorithm asymmetrical. An explanation from a tutorial by Lode suggested to instead use exact ratios with integers to determine how far along the x or y length of a square we were, guaranteeing an ending at exactly the edge or middle of the target square every time. (Vandevenne) The end result solved all the above problems but runs up to 8 times slower than normal raycasting for results that fail in worst cases (bring surrounded by walls on all sides), though the smaller radii of the average entity calling raycast checks makes it reasonable enough to never be noticed.

3.12.4. Invisibility

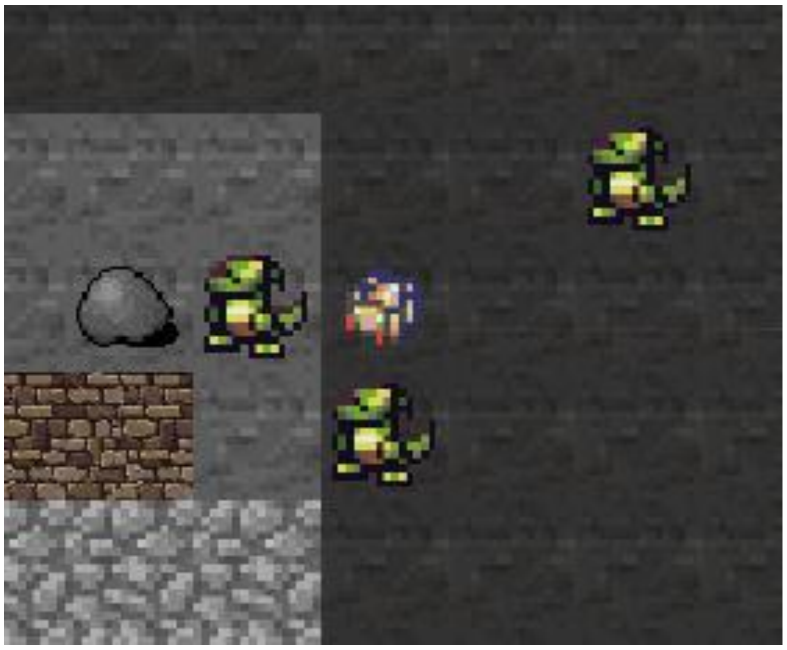


Figure 25. A Seeker spider has faded itself, becoming invisible and able to sneak next to the kobold miners without causing alarm. Source: Screen capture.

A unit can be effected by invisibility status effects that render them untraceable to most normal methods of being noticed, with three exceptions: they're made to be revealed by a status effect that marks them as always visible to any viewer, the unit trying to see them has a 'truesight' effect which lets see invisible units, or the unit trying to see them looks like an ally, as invisible units reveal their presence to everyone they think is on their side. Items work differently, being always visible if they're on the ground but share their owner's visibility if being carried.

Invisibility goes hand in hand with fog in the visibility theme, and like the fog efforts were designed to make it feel as intuitive as possible. One such correction was pathing: if one can't see something in the way, what happens when they try to move through it? Being told you can't move to a location without being told why is clearly a design flaw, but a wall made to be invisible shouldn't just stop having the properties it did before. Figure 25 illustrates an example of a seeker-spider invisibly scouting out potential victims.

Common sense dictated that the mover would accidentally bump into the invisible blocker and reveal its location, which leads to units being able to 'shake' each other. Any unit that does something or is affected by something that would realistically cause them to become unbalanced or disrupted in some way (such as being bumped into, attacking, casting a spell, being attacked, etc.) makes them react by being shaken for some amount proportional to how disturbing the force was. For every invisibility-related effect, any amount of shaking is enough to cancel it and remove the invisibility, fixing the invisible-blocker problem and opening up many other methods of interaction. Shaking can be used in other ways as well: large enough shakes, requiring intentionally disruptive abilities, can interrupt multi-turn (aka channeling) tasks or even knock away the victim.

3.13. Memories

All units have a list of memories about other units they've encountered in their travels. If a unit can see a square and another unit is already there or enters that square later, the seeing unit remembers where the unit is, and once visibility between them is lost, remembers the time and place where they were last seen. Figures 26-28 illustrate how these mechanics are displayed onscreen.

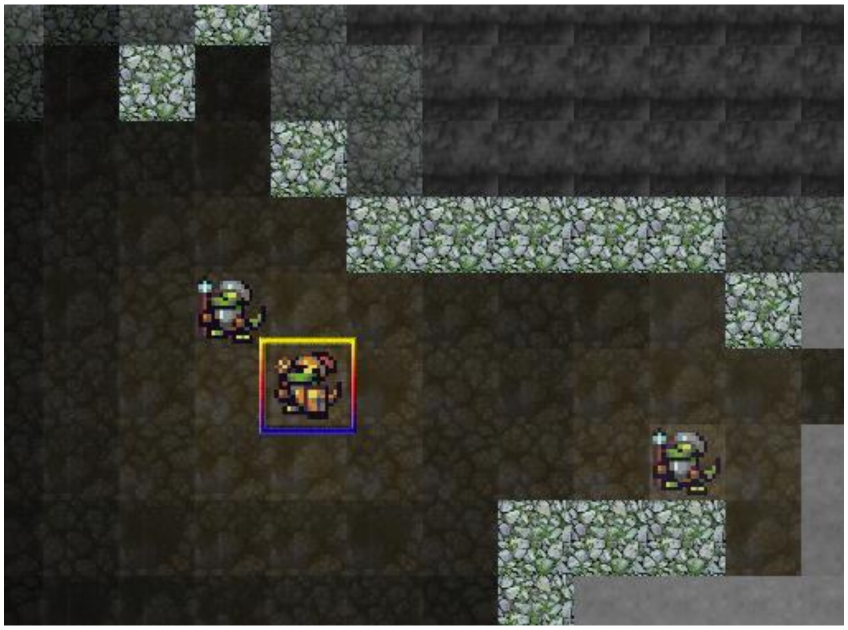


Figure 26. A player notices an allied soldier (far right) with a lantern on. They are actively watching the soldier, but still keep a memory of them. Source: Screen capture.

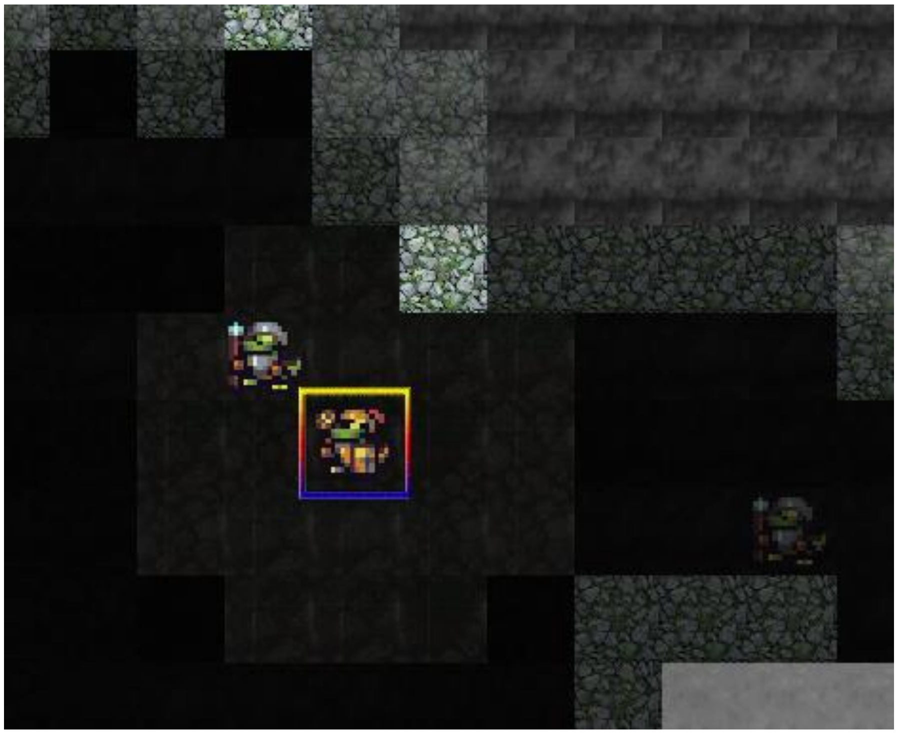


Figure 27. When the lights go out, the soldier's last known location is remembered. Source: Screen capture.

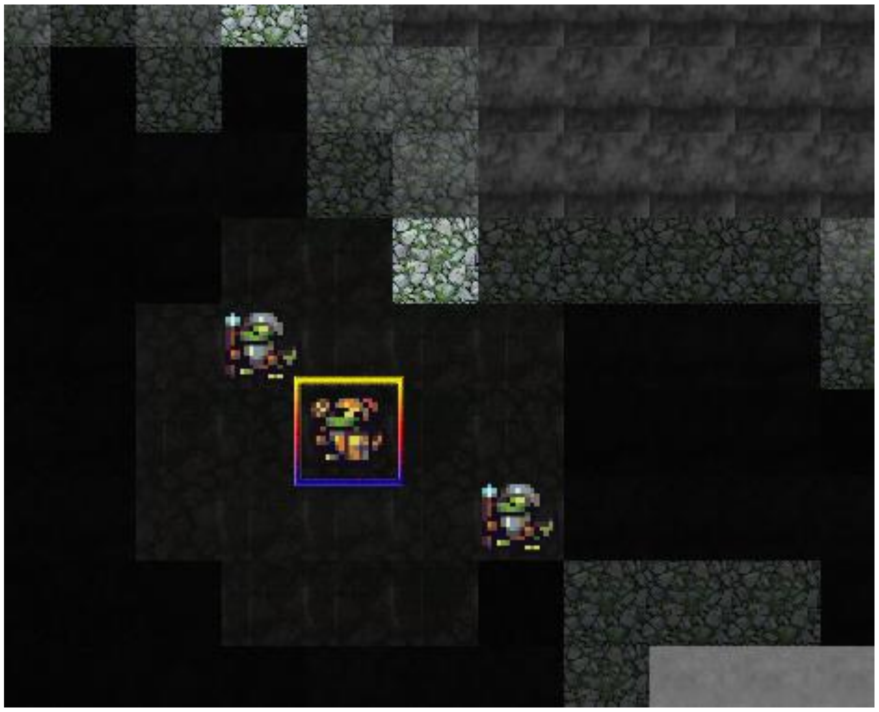


Figure 28. When the soldier steps into view again, their old position is updated to reflect the new information. Source: Screen capture.

If a unit witnesses another's death, the memory is updated to note that their death occurred there instead of just a sighting, or if someone attempted to look for a unit that was somewhere previously but wandered off, will update that memory to be a reminder that they're not there anymore. Similarly, units will remember all the squares they've seen at least once to tell the difference between completely unexplored areas or squares they've been to but didn't have anyone there.

3.13.1. Initial conception

Memories are what came of trying to solve the problem of scouts having to return with information, and are arguably the most interesting / definitive feature of the game. The first, admittedly easier version was to simply consider territory in terms of zones: friendly and not friendly. For a standard RTS, consider anywhere nearby one's base or central HQ as the 'friendly' territory that's almost always completely in view with no hidden corners, and anywhere outside as being neutral or hostile territory. A scout, or anyone else, leaving the friendly zone would have their discoveries lay dormant until they returned to friendly territory, whereupon it would instantly update everyone and the player to the map outside as they last saw it. This brought to question how the scout, or anyone else, would be controlled outside of friendly territory if they couldn't be seen, or if this kind of 'echolocation' vision of periodic map updates would be enjoyable to a player back when the game was angling towards an abstract-player god-like angle. Figures 29-32 (below) illustrate how the memory mechanics are displayed to a player, alone and when interacting with an ally that shares its memory.

Further brainstorming regarding zones continued to bring up cases of them being too abstract and abusable a concept that was sure to fail in unrealistic ways, with the main benefit often only making the game seem more comparable to a standard RTS. The idea of using zones was scrapped, and instead more drastic measures were thought up: Instead of scouts being independent when they left the base, instead what if everyone was independent at all times? What if not only scouts had to report their findings, but the resource gatherers and hunters and soldiers as well?

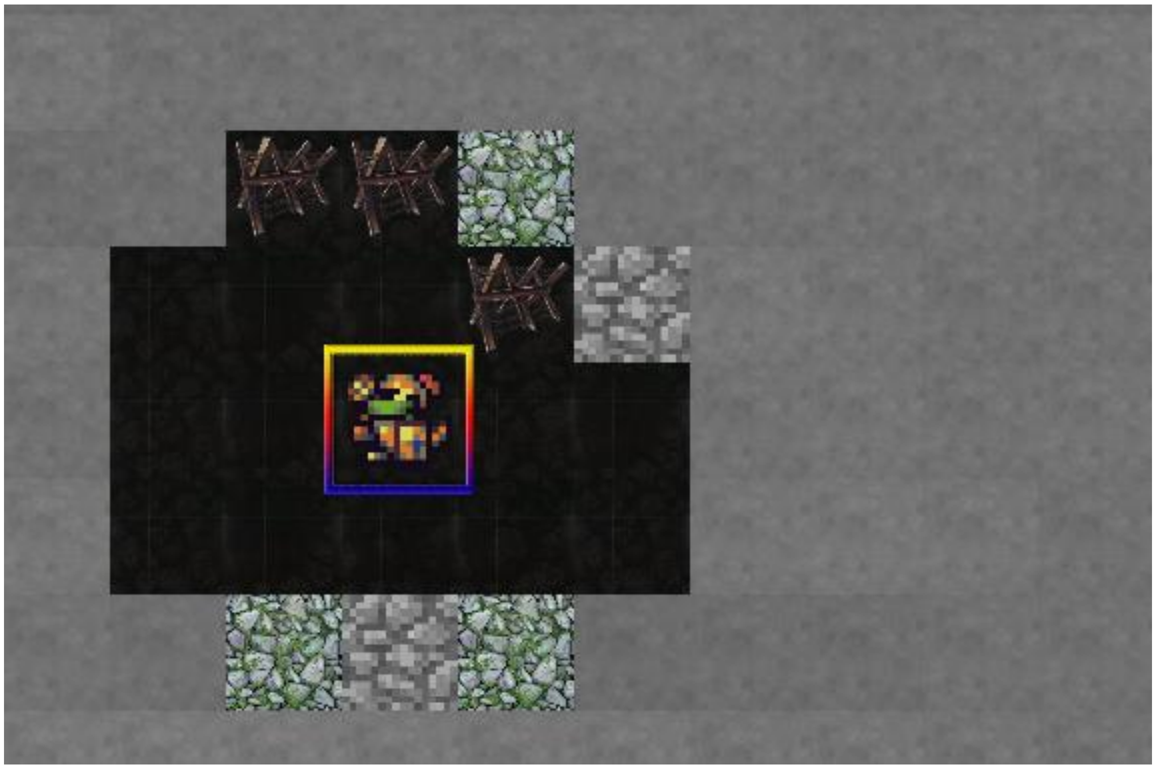


Figure 29. A freshly-spawned player in the dark, knowing only what they can see. Source: Screen capture.

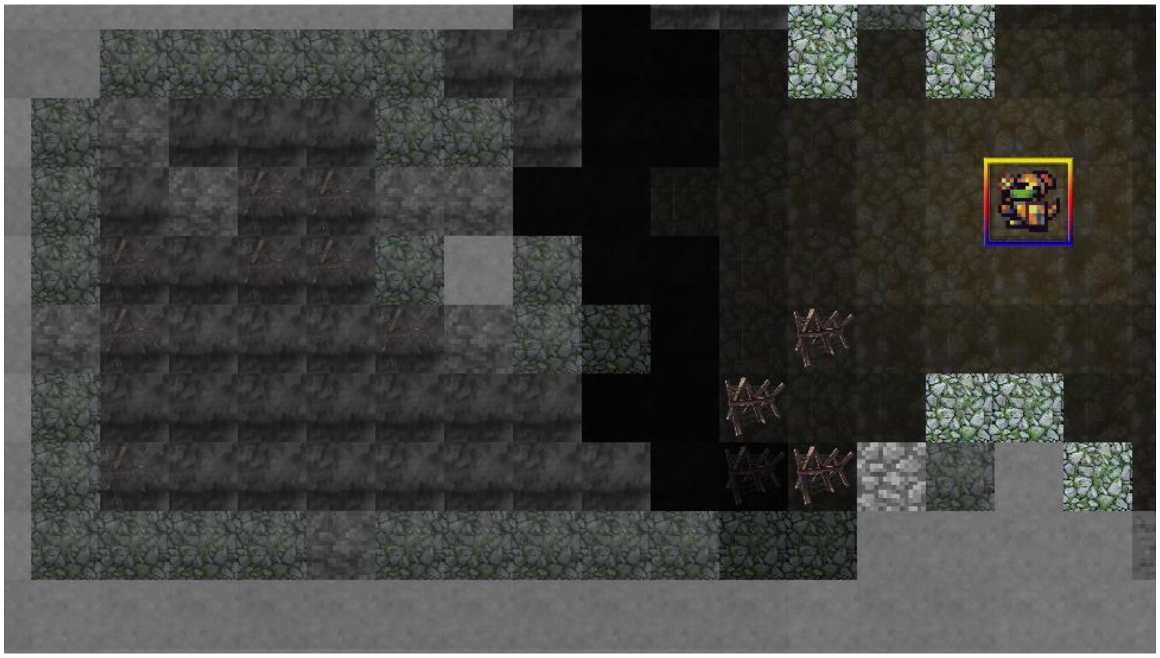


Figure 30. After a bit of exploring, the player still remembers where everything they saw last was. Source: Screen capture.

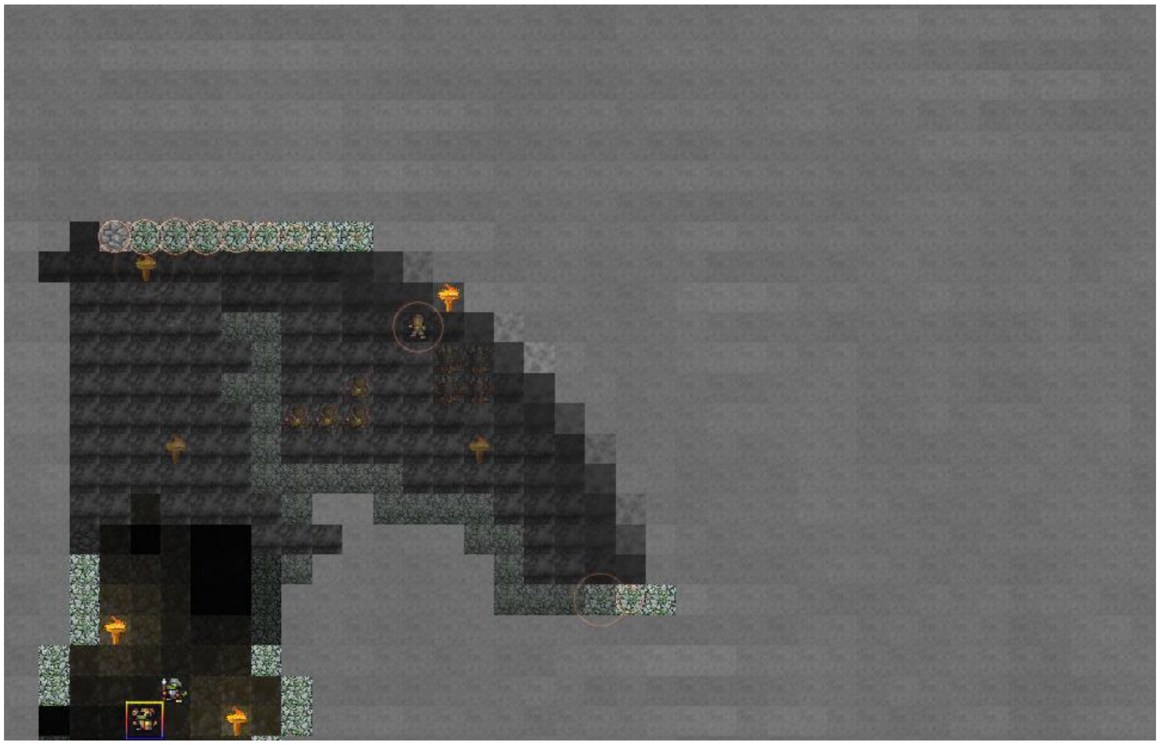


Figure 31. Upon meeting an allied soldier, the player communicates with them, showing an animation of any information they have to give. Source: Screen capture.

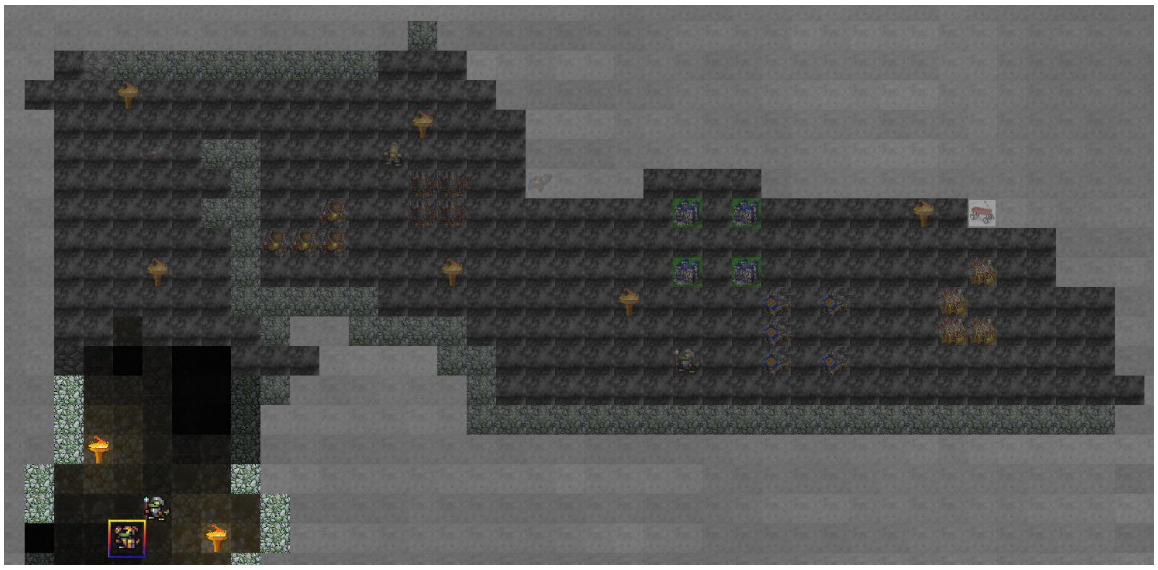


Figure 32. Even though the player has never been to the newly revealed area, they still acquired memories of it from the soldier they talked with. Things may have changed in the meantime, but this was the state of the world last he saw. Source: Screen capture.

As most units only care about one aspect of the game world at a time (where walls are for miners, where bats are for hunters, etc.) they could all try to keep up to date with the things they cared about at the base, in turn necessitating a meeting ground and way of exchanging information. Everyone would keep track of their own map, even if they were doing things together in a group, with no shared vision at all.

3.13.2. From concepts to concrete

Initially memories were much simpler, being the location for concepts that each race considered differently. All units would have a memory of their last known 'mining' spot, which was always the last mine-able wall they saw, and when they noticed a new one would update their memory to it instead. Units could then talk to each other, giving a new location if one didn't have any yet, which was usually done at their race's capital that would almost always have something for everyone. Heading to a memorized location only to find nothing there would mark that memory as outdated, and anyone attempting to trade that exact memory with the same unit would become updated in turn. This meant everyone could only retain one location per concept at a time, which would eventually fail for two reasons.

The first and foremost was that only one location at a time being memorized meant units were very short-sighted involving multiple things in an area, especially things that could move. Bats are a hunter's target, being stored in their huntingGround memory, but chasing a single bat that ran away from its herd and killing it would result in a path to its last known hunting ground, i.e., the bat they just killed, declaring it out of date then having no idea where to go from there. Cases like these happened often, frequently causing miners to end work halfway through a large chunk of walls or for fighters to call off the hunt early in situations that didn't make sense without exact knowledge of why their memories seemed so short-lived.

The second reason was the concepts being too abstract: different races had different definitions for what was hunt-able, mine-able, friend or foe, etc., and any communication between them became conflicted with 'translation issues.' Hunters declared their most recent bat sighting as their hunting memory, while spiders who were picking off anyone with the ‘prey’ tag considered goblin workers, even those hunters, as their quarry they stored in their huntingGround memory. If interrogation was ever to be added, or any attempt to see how other races thought and remembered, this was going to be a huge issue. Concepts weren’t encompassing enough; Units needed to remember everything they saw, even if it wasn’t completely relevant to them at the time.

3.13.3. Detailed memories

Thus, memories became a list of units seen. A ‘memory’ was now a new class entirely, containing a number of variables to specify a bit more about the specific memory associated with them. The location of the target being remembered remained, and in the case of large units occupying multiple squares at once only their center location was stored. A last-seen-status value replaced the system for checking out-of-date memories, being able to differentiate a ‘memory’ of something currently being viewed, a memory the unit still believes is there but is out of sight, a memory that something wasn’t where it was last searched for, and a memory of something believed to have died. The difference between ‘dead’ and ‘missing’ allowed for complete closure as to if something should still be considered worth searching around for, and the last-seen-status value became a deciding factor in whose memories transferred over to whom when trading information.

Knowing how long ago a memory was created was also important, recorded in an ‘age’ value, and in the old cooldown-based system every memory would increase in age along with all the other units when an update was handled. When the newer tick system was implemented, age was replaced by recording the last tick the memory was updated and didn’t need to be constantly aged alongside units anymore. The last threat level the remembered-target was seen at was also recorded: captured / subdued enemies were no longer tracked down, discovered to be harmless, ignored then tracked down again the moment they were out of sight. The unit who first created the memory is recorded as well, which helps to keep track of who started causing problems in the debugger but didn't have any impact on gameplay, though it could conceivably be used to track the source of a blatant liar.

3.13.4. Trading info

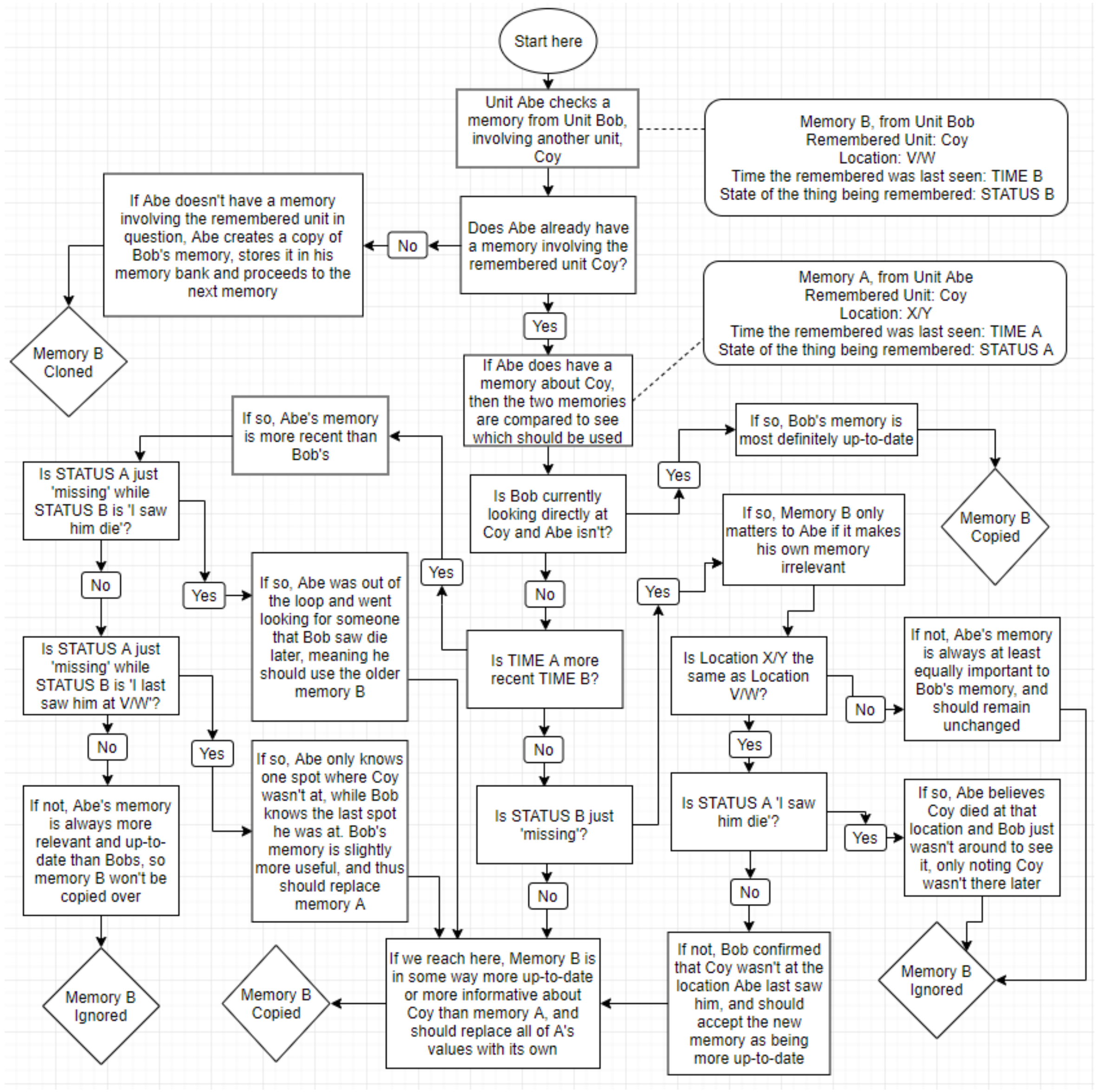

Start here
Unit Abe checks a memory from Unit Bob, involving another unit, Coy
Memory B, from Unit Bob
Remembered Unit: Coy
Location: V/W
Time the remembered was last seen: TIME B
State of the thing being remembered: STATUS B
Does Abe already have a memory involving the remembered unit Coy?
No
If Abe doesn't have a memory involving the remembered unit in question, Abe creates a copy of Bob's memory, stores it in his memory bank and proceeds to the next memory
Memory B Cloned
Yes
Memory A, from Unit Abe
Remembered Unit: Coy
Location: X/Y
Time the remembered was last seen: TIME A
State of the thing being remembered: STATUS A
If Abe does have a memory about Coy, then the two memories are compared to see which should be used
Is Bob currently looking directly at Coy and Abe isn't?
Yes
If so, Bob's memory is most definitely up-to-date
Memory B Copied
No
Is TIME A more recent TIME B?
Yes
If so, Abe's memory is more recent than Bob's
Is STATUS A just 'missing' while STATUS B is 'I saw him die'?
Yes
If so, Abe was out of the loop and went looking for someone that Bob saw die later, meaning he should use the older memory B
No
Is STATUS A just 'missing' while STATUS B is 'I last saw him at V/W'?
Yes
If so, Abe only knows one spot where Coy wasn't at, while Bob knows the last spot he was at. Bob's memory is slightly more useful, and thus should replace memory A
No
If not, Abe's memory is always more relevant and up-to-date than Bobs, so memory B won't be copied over
Memory B Ignored
No
Is STATUS B just 'missing'?
Yes
If so, Memory B only matters to Abe if it makes his own memory irrelevant
Is Location X/Y the same as Location V/W?
No
If not, Abe's memory is always at least equally important to Bob's memory, and should remain unchanged
Yes
Is STATUS A 'I saw him die'?
Yes
If so, Abe believes Coy died at that location and Bob just wasn't around to see it, only noting Coy wasn't there later
Memory B Ignored
No
If not, Bob confirmed that Coy wasn't at the location Abe last saw him, and should accept the new memory as being more up-to-date
No
If we reach here, Memory B is in some way more up-to-date or more informative about Coy than memory A, and should replace all of A's values with its own
Memory B Copied

Figure 33. A flowchart showing how memories are selected for copying, replacement, or ignored during the tradeInfo process.

The transfer of memories between units was meant to be an intuitive process, though the result ended up looking extremely complex, as suggested by Figure 33 (above). Two allied adjacent units can trade information with each other through direct contact, which essentially takes all the relevant memories from one unit and gives them to the other then repeats with the units reversed. The unit being searched iterates through all of its memories: if a memory is completely new, the receiver creates its own copy of it, but if they both contain a memory of the same unit each are compared to check for which one gets updated.

Memories currently being seen have the highest priority, always replacing the other as they’re seen as always accurate (“I saw Dave a while ago, but you’re looking at him right now, so I’ll trust you’re correct”).

Next, if the receiver’s memory is more recent but only knows the unit wasn’t there at the last check, their memory could be replaced if the giver either knew for sure it died (“I didn’t see him at the docks a minute ago, but Fred saw him die a week ago”) or was seen at another spot before (“I didn’t see him at the docks a minute ago, but Fred saw him at the carnival two minutes ago”).

If the giver’s memory is just that they haven’t seen them at their location, and the receiver’s memory involves anywhere else, they ignore the memory and nothing gets updated (“I didn’t see Jeff at the docks, and he didn’t see him at the carnival, so who cares?”).

If none of these edge-cases occur, the last check is that the other memory is more recent that the receiver’s, and if so all of giver’s statistics are transferred over to the receiver. The process repeats until all the giver’s memories are checked, at which point the receiver is fully up to date with everything the giver knows and can trade positions. The process would require a bit less iteration if the transfer was mutual, which it initially was for a time, but having the process be one-sided allows for interrogations where only one unit extracts information from another.

3.13.5. Individual use

More importantly that having or trading memories was being able to use them for something. Whenever a task called for searching for something a unit may know about, that Unit's memories were scanned through for the closest relevant thing to become the target. One of the first efficiency changes was to add a list of watched units similar to memories, to be used as a short circuit check before shifting through the entire library of every remembered Unit, and also to add some intuitiveness by always going for things in sight even if other memories were 'closer' but not in view at the time (A bird in the hand is worth two in the bush). Filters were added to check for things like being within a certain radius, the difference between being an ally vs not an enemy vs neutral, looking only for things worth trying to reach, preferring melee targets over ranged, etc. Iterating through memories this way allow every unit to base their decisions off their own map state at any given time.

3.13.6. Garbage collection

The thought of garbage collection for 'dead' memories came up, with bad results after a large amount of work and testing. The first iteration involved an additional 'intelligence' value for units that determined how long their memories could last before expiring. Regular units could remember things for 100 turns, capitals and such could remember for up to 10,000 turns, while critters like bats could only remember for 5 turns or so basically making them forget everything the moment they walked out of sight. There was still no limit to the amount of memories one could hold, only their duration if left without updating via seeing in person or being told about them. This would allow relevant memories to keep getting refreshed when they were traded at the capital hub, while also allowing for a new paradigm of manipulating intelligence to make units remember more or less with specific effects.

Both intentions failed: expiring memories never turned out well, causing countless pathing issues in places that weren't 'recently' explored while not reducing the size of memory arrays by any significant amount. Modifying intelligence to be greater had no apparent effect from a player’s point of view, as memory length is usually requires a very long-term investment to notice any changes, while reducing intelligence certainly made units appear more stupid, but usually just through erratic pathing and / or forgetting where their main home was and just standing around or walking in circles trying to find it. These inevitable higher / lower bounds wouldn’t be very fun or interactive from a player’s point of view, and manipulating a player’s memory in that way was simply unthinkably horrible, so the idea was discarded.

The only other point of garbage collection afterwards was the need to go to a memory’s last known location to confirm the unit in question wasn’t there anymore, which was causing buggy issues and was somewhat unrealistic (A miner would have to stand in the exact spot a wall used to be to realize it wasn’t there anymore, for example). Checking all memories on every update for anything that should be in view and making sure that they were was the solution, setting their status to ‘missing’ if not found, though this process is horribly inefficient CPU-wise.

3.13.7. Player-specific adjustments

A player’s memories are the most important, and required a bit more detail to make things more clear and intuitive. These are the only memories that need to show up on the screen, and as discussed in the rendering section this can be a bit more difficult than first imagined. Multiple units at the same location had to be concatenated to a single one with a box symbolizing more were present, and the difference between an out-of-sight memory vs a present viewed subject needed to be crystal clear. This is the main cause of the constant missing memory checks mentioned before, as having that in meant the visible squares could be reserved for only ‘real’ present units while out-of-sight squares in the fog were purely memories. In fact, players have no way of telling what their ‘missing’ or ‘dead’ memories are, as they don’t show up on their screen, though they still transfer the memories and information when they trade info with allies.

Trading info in particular needed to be informative, as any changes should be noticeable while the overall map may look the same. This led to a ‘sonar’-like reveal animation inspired by the map from *Darkest Dungeon* (Red Hook, 2016), where any changes are sorted in distance order and highlighted as a circle expands outward from one’s character. Old memories are cleared and moved to their new locations, while new ones flash out as the circle goes over them to catch the player’s attention.

The player’s point of view also necessitated an additional thing to keep track of: squares that were explored at least once, regardless of whether they had units or not, as there was no way to tell previously what had been explored unless a unit was there. Now all units keep track of every square they’ve cast their ‘watched’ raycast check to, mixing the results when they trade info and giving the player the sum along with everything else when they meet up, so empty tiles reduce the fog from its ‘heavy’ unexplored amount to a negligible ‘explored but currently unseen’ amount as a visual indicator.

3.14. AI

The flexibility granted by how tasks are created or removed so easily means that AI should be considered much more the sum of its parts than any individual unit. The intricacies of how things interact with each other and react to their environment combined with all the ways their knowledge of the world is constantly being limited make it difficult to pin down specifics about the upper or lower limits of their behavior.

3.14.1. Independent AI

An important note about the nature of the game is that there is never any top-level controller looking on and commanding from above: all interaction between entities in the game is direct, with no abstract third party giving orders as in a typical RTS. Each individual in the game has their own set of motivations and beliefs about what they can and should be doing at any given moment, only performing whatever behaviors they start with or acquire along the way. Even when given commands, the individual following them is completely unaware of how what they are doing affects the world for good or bad. No exceptions exist in this regard, but there exist some units that do care about the macro-level economy and overall well-being of their group within their limited scope of knowledge. A player would be the prime example of this, able to make educated guesses about the state of their team compared to the enemy given what they know, and react by giving orders accordingly, and from the AI we have commanders and capitals acting to emulate a human's behavior as much as possible.

3.14.2. Capital AI

Capitals are the central hub of a team / race's civilization, where everyone heads to for information about what needs doing and to provide their findings to the hivemind at large. Capitals, being stationary buildings, don't do anything on their turn normally and are activated and updated every time a unit interacts with them, progressing through their thought process depending on their world state after trading information. All capitals, as a general following between all possible races, have a 'need' queue for things they think need to be done with top priority, a 'greed' queue for luxury tasks that are helpful but not absolutely necessary, a 'resupply' queue to ensure independent storage facilities are occasionally emptied and brought to a more centralized location, a 'build' queue for whatever construction or repair work needs to be done, and a 'check' queue to check in with all known buildings now and then to make sure they're still standing.

When a unit interacts with a capital and hands over any extra inventory items (unusable supplies like ores or raw meat), the capital quickly checks its memories and inventory to see what it thinks is lacking, and if anything is on the queue, and depending on the type of unit they're given a new task that will fulfill that need somehow. A lack of some type of item usually involves a delivery to the building that can craft it with the necessary supplies, a recent report of fighters going missing for long periods of time may involve a message to the training grounds asking for replacements, all usually amounting to fetch / delivery / check-up quests that are swiftly accomplished with the dutiful unit then heading back to check for more chores and repeat the process.

The details of tasks given out are dependent on the unit receiving them: carts are usually only given delivery tasks to and from storage areas, most basic units receive things related to their normal duties, while some only come to a capital to update their own knowledge of the world. Commander-type units are among those, treating the capital like a glorified messaging board to find the next area of interest that requires their attention. Currently, this amounts to finding any as-yet undefeated enemies, waiting around the capital for military-grade units to recruit, overriding their default tasks with an order to follow them to war and waiting for a reasonable sized army to head out and fight anything that looks at them funny along the way before heading back to refuel. The combination of a capital and commander was meant to simulate a typical player's thought process, with any additional behaviors being added on a race-by-race basis based on their strengths, weaknesses or other necessities.

3.14.3. AI: Too dependent?

A difficult opinion lies in the question of how smart is too smart, or how dumb is too dumb, when dealing with independent AI. Players coming in from other genres are used to bases that run on auto-pilot if left untouched and units that only do the last thing they're told before waiting for further instruction. The independent nature of the memory and task system necessitates a smarter individual, but smarter is a very relative term: it may seem smart for a unit in possession of a sword to swap to it and fight in the face of an enemy in general, but there are often scenarios where even the time spend equipping the sword is better spent running away to safety 5 feet away.

Not every unit can be engrained with the intelligence of a player, yet we've developed the game around limiting micromanagement to a minimum. This draws the difference between tactics and strategy: a player needs to learn the behaviors of all their allies, not expecting them to always be perfectly rational agents in their field, but more of a tool meant for a specific part of gameplay: fighters charge enemies, archers stay in the back, workers flee from enemies but rush towards walls, etc. Everything isn't given player-level intelligence for a reason: AI in every game is meant to be predictable and flawed, with the challenge and fun of a game being the efforts to abuse and overcome it, and exploit it fully when it's on your side.

3.14.4. AI: Too independent?

On the other hand, the game's anthill-simulator roots are apparent in its multi-dimensional design. Even if autonomous entities behave imperfectly on a turn-by-turn basis, we wouldn't want them smart enough to complete the whole game for the player. Most of the automation mentioned previously with commanders and capitals is on the part of the commander, which is entirely replaced by the player, and the capital parts are manually disabled when a player is on the capital's team. This doesn't stop a capital from giving out basic commands, only limiting their task-giving systems to whatever the player assigns them to do. The player can queue up items and units to be created / trained, and the capital will handle the rest. The player isn't prevented from manually redirecting units to where the capital would tell them to go anyway, or carrying and delivering supplies himself, but such automation is naturally left to the 'system' similar to mining.

If units behaving too intelligently for micro-intensive behavior like kiting or combat is a problem, then only their attributes need to change: moving faster or slower, doing more or less damage, fleeing at different health values. Anything can be tweaked to strike up a good balance between fair fights and smart ones.

3.15. Summary

Trick of the Light's overall experience goal is to have the player in an environment they're familiar with from other genres, but training a mental muscle that rarely gets touched in other games considering the themes of imperfect information. Most of the player-side gameplay is highly correlated with that of the roguelike and RPS genres, having numerous options to interact with the world around them (mostly related to combat) and being able to personalize themselves and followers with weapons and armor. The macro-level gameplay requires strategic thinking in line with a standard RTS, having base-management and Unit-upkeep as primary concerns, though often a fire-and-forget one that involves queueing up and waiting for the results. The goals and flow of the challenges are in the same style as a classic adventure, leading the player along an interesting narrative that puts them at the center of the story.

Blizzard's *Warcraft 3* (2002), though an RTS, would be considered the primary inspiration for how the game turned out in this regard, having all the elements in the same way described above, but also being the primary offender in the first question we sought an answer too at the beginning of this project about the sacrificial scout. Trick of the Light's step into the territory of imperfect information should cast a shadow of doubt about how a player typically trusts their own in-game mental state. Not everything their character sees is real, nor are all the things they're told are true. Not everything that occurs within their territory will be relayed to them, and the lack of information should start to become just as telling as receiving it in some cases. Deductive reasoning and a slight sense of paranoia are absolutely the critical separation from the aforementioned genres, enunciated through the more immersive and realistic themes even when the actual characters are goblins and ghouls.

The extra visibility mechanics all have the same purpose as normal fog of war, in that they limit character's vision in a somewhat realistic manner to reduce the amount of 'perfect' information they have. Even within their own sight radius things are constantly being hidden around blockages, in the darkness or simply invisible to the naked eye, each adding an element of uncertainty to the only direct source of vision they have in the world.

Fog of war at its core is intended to emulate the real-life property of never being omnipotent about a situation: there are almost always unknowns that need to be accounted for, the allied side just as much as the enemy, and implementing so many things involving one's personal vision radius is a way of suggesting that even depending on everything one can see may be dangerous without taking certain preparations or being overly thorough, which is rarely a luxury that can be afforded.

4. Technical Design

The code of Trick of the Light has gone through many re-bases and language changes, learning many common practices and general formatting techniques. The end result is always the most critical goal, but the processes put into the engine are ultimately the ones that shape the flow of the game most of all.

4.1. The World class

Starting the game initializes the World class, the main hub of the engine where all decisions are resolved and effects ultimately applied. The World is a static class, meaning there's only ever one instance of it at any given time. That instance is always called by anything interacting with the world to ensure everything is taking place in the same 'universe'.

The initialization process starts with reading a map file and starting to print out units and items at their designated locations, but spawning them in the normal way would cause problems: Units being placed in sequence with walls would see areas and things they weren't supposed to if everything spawned in at once, so the normal creation process is separated into chunks of placement, glow-casting, start-reacting, vision casting and then hard refreshing, normally done all at once when a new entity is created. The ‘refresh’ function mentioned above is a Unit-only function that ensures all non-internal sources affecting the unit are checked again, with the option to make it a ‘hard’ reset that will recalculate everything the unit is looking at as well as everything which may be looking at them. This is mostly necessary for large changes, such as a sight-blocking wall becoming invisible and letting everyone attempt to check if their sight radius was updated.

4.2. Maps: A tailored experience

Map generation was a key consideration for how the game would be played, and what kind of game experience would be created. *Trick of the Light’s* presentation is visually similar to a roguelike game. Its maps could also have been produced a similar way: by procedural generation. Doing this would have introduced additional elements of exploration and uncertainty that would align with the experience we wanted to produce. However, the core objective of *Trick of the Light* was to see how well players would comprehend and react to its memory and fog mechanics, not to produce a highly replayable game. Adding geographic randomness makes no sense in an experiment that is only expected to be played once or twice.

4.2.1. Campaign formula

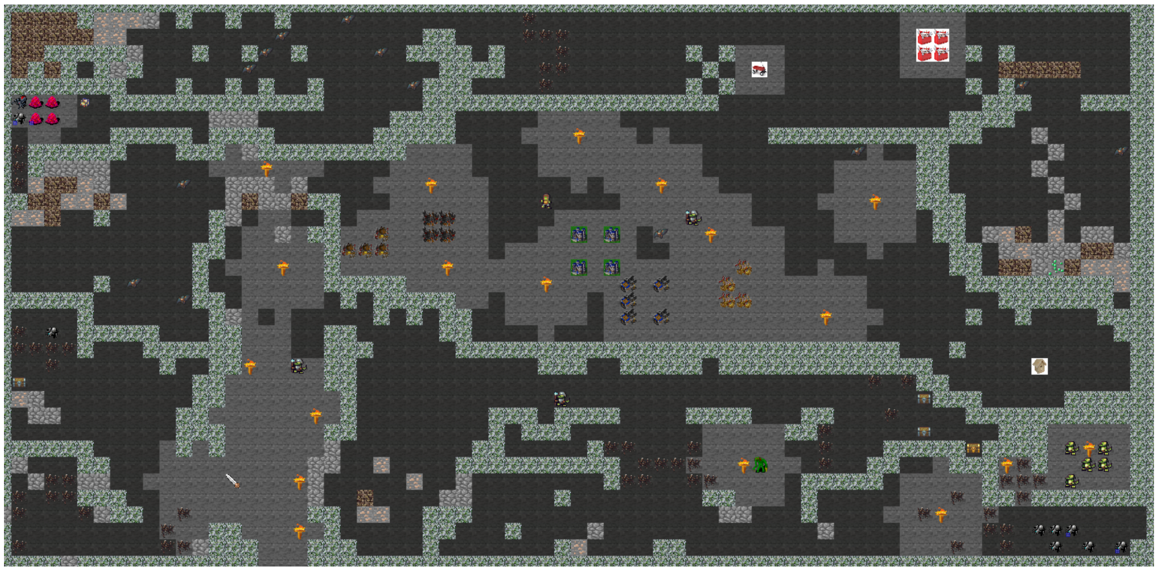
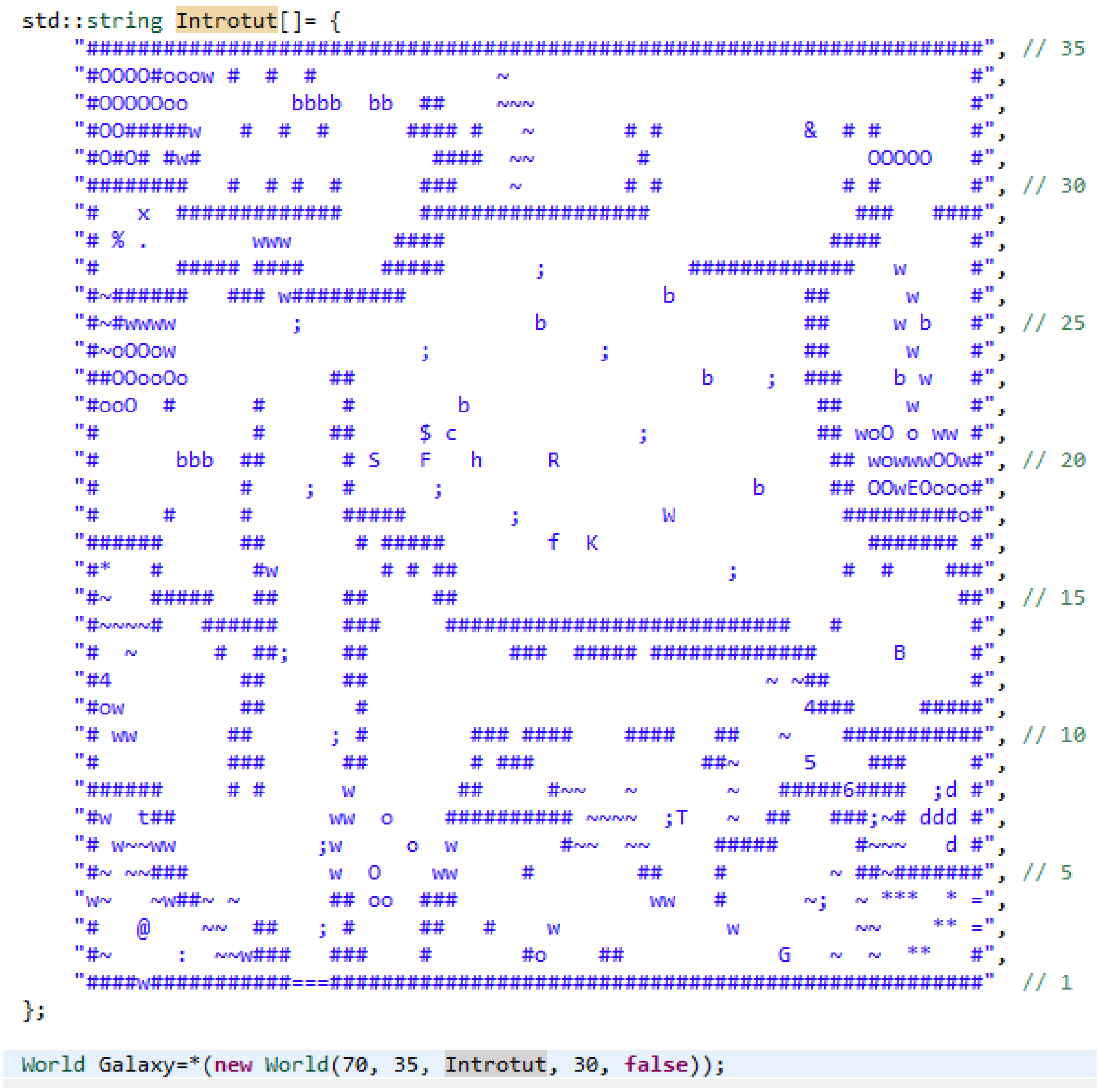


Figure 34. A full view of the tutorial level. Source: Screen capture.

The decision not to employ randomization positioned *Trick of the Light* squarely within the RTS tradition of single-player campaigns. A 'campaign' is a series of levels of increasing difficulty, usually introducing a single unit type and/or game mechanic on each level. It is essentially an extended, well-integrated game tutorial, which is exactly what we wanted to ensure that the elements of the game we wanted explained would be taught to every player the same way. Campaigns also allows for scripted events and one-time gimmicks that won't be reproduced elsewhere in the game, such as dialogue between characters or spawning / despawning items and units after certain conditions are met, which help to create situations and storylines that make learning as intuitive as possible.

Trick of the Light was initially planned to have its own campaign in the same format, showing off all mechanics in chunks, but we quickly realized the scope of a full campaign would scare away potential playtesters. At 10-20 minutes per level, anyone who didn't like the initial mechanics shown were unlikely to proceed through the rest of the game to learn the rest, which very early unofficial testing undoubtedly confirmed. It was decided to compact the most iconic mechanics into a single level that would introduce all of them in sequence. The most important part of testing was to see if testers could understand the core vision, fog and memory systems, which a single level could provide. Figure 34 (above) shows the layout of the tutorial map used for initial playtesting. Two additional levels were prototyped and partially implemented. Their layouts are displayed in Appendix G.

4.2.2. Generation



```
std::string Introtut[]= {
    "######################################################################", // 35
    "#OOOO#ooow #  #  #              ~                                    #",
    "#OOOOOoo        bbbb  bb  ##    ~~~                                  #",
    "#OO#####w   #  #  #      #### #   ~       # #           &  # #       #",
    "#O#O# #w#                  ####  ~~        #                 OOOOO   #",
    "########   #  # #  #      ###    ~        # #              # #       #", // 30
    "#   x  #############      ##################                ###   ####",
    "# % .        www        ####                              ####       #",
    "#      ##### ####      #####       ;           #############   w     #",
    "#~######   ### w#########                    b          ##      w    #",
    "#~#wwww         ;                  b                    ##     w b   #", // 25
    "#~oOOow                   ;             ;               ##      w    #",
    "##OOooOo           ##                           b    ;  ###    b w   #",
    "#ooO  #      #      #        b                           ##     w    #",
    "#            #     ##     $ c              ;             ## woO o ww #",
    "#      bbb  ##      # S   F   h     R                     ## wowwwOOw#", // 20
    "#           #    ;  #      ;                        b     ## OOwEOooo#",
    "#     #     #       #####        ;           W             #########o#",
    "######      ##       # #####        f  K                     ####### #",
    "#*   #       #w        # # ##                     ;        #  #    ###",
    "#~   #####   ##     ##     ##                                       ##", // 15
    "#~~~~#   ######     ###     ###########################   #          #",
    "#  ~      #  ##;    ##           ###  ##### #############      B     #",
    "#4          ##      ##                               ~ ~##           #",
    "#ow         ##       #                                  4###     #####",
    "# ww       ##      ; #        ### ####    ####   ##   ~    ###########", // 10
    "#          ###      ##        # ###             ##~     5    ###     #",
    "######     # #      w        ##     #~~   ~       ~   #####6####  ;d #",
    "#w  t##            ww  o    ########## ~~~~  ;T   ~  ##   ###;~# ddd #",
    "# w~~ww           ;w     o  w        #~~  ~~     #####      #~~~   d #",
    "#~ ~~###           w  O    ww     #        ##    #        ~ ##~#######", // 5
    "w~   ~w##~ ~       ## oo  ###               ww   #      ~;  ~ ***  * =",
    "#   @    ~~  ##   ; #     ##   #    w             w         ~~    ** =",
    "#~     :  ~~w###   ###    #       #o    ##            G   ~  ~  **   #",
    "####w###########===###################################################"  // 1
};

World Galaxy=*(new World(70, 35, Introtut, 30, false));
```

Figure 35. What the level looks like in ASCII form. Every character symbolizes what character goes where, including some special scripted characters that have additional tasks and such manually added to them on creation. Source: Screen capture.

Map creation is done within the ‘main.h’ class, taking in a manually-made 2D string array filled with characters that represent the units / items to be placed (see Figure 35). The world is then created by manually scanning the array and putting things where they look like they should be on the ASCII ‘map.’ While obviously not the most sophisticated choice, it’s been working since the start of the project with no major reason to upgrade to anything else thus far. Offloading the mapmaking process to an outside script would make it editable without having to rebuild, but most changes for testing or debugging rarely require that much fine-tuning after a general change, and the rebuild process for changing a string in main is negligible.

If we were to expand the game further, mapmaking would definitely need an improvement at some point, with some early inspection being done about how the Tiled map editor software could be integrated. (Tiled) Tiled is intended specifically for 2D grid-based map systems like we have in *Trick of the Light*, but why even go that far to add an outside source to the game when the engine could handle it quite easily by itself? An in-game editor would only require a menu system for selecting what to add where, with saving and loading being simple read-write from a text-log. A few playtesters that experimented with the debug-view of the game were genuinely surprised the feature wasn’t already in, believing they missed a button somewhere that would have everything they needed ready to go. The feature could easily have been put in at the time if we thought anyone would be interested.

4.3. Formatting practices

Formatting and general code style hasn’t been a serious problem due to the one-man development team so far, but the ‘so far’ aspect being subject to change led to some common-sense minimal standards. The spacing of indices and such are consistent throughout the whole project, comments are available where complex or non-obvious decisions are made, return types and input values are listed at the front of every function, all enough so that someone reading things for the first time would know how things worked if not the order they should start looking.

The most populated class types are units, items, status effects and tasks, each containing templates to easily generate new ones. Further templates are available for certain ‘genres’ of classes, like a kobold type unit or an item-producing building, but frequently require a clone of their related tasks as well to change any of their standard behaviors: the unit class only defines what the unit is like statistics-wise, while all of their actual activity comes from whatever tasks they’re initialized with during creation.

4.3.1. Class-centric practices

An ongoing problem is determining what functions should be put in World vs what should remain within one of the Living subclasses, such as spawning in new entities or handling interactions or reactions between them. The general rule is that if something should remain constant throughout all possible instances or subclasses it belongs as close to World as possible, while functions that have even a slight possibility of being overwritten for some specific use case should belong in one of the living subclasses to be modified at the specific unit / item / Status effect level when need be. For example, an entity being moved from point A to B should always result in a few things, like being transferred between squares and handling any glow / vision changes that arise because of the movement, and thus became part of the World’s moveItem/unit functions instead of having a unit/item-based movement function. The attempt to walk, however, may be dependent on the Unit’s class: imagine if spiders were able to walk through squares with webs in them, regardless of what other units were on the square, or a type of magical golem that was strictly forbidden from trying to walk too far away from its power source. Cases like those are why functions related to ‘trying’ to move are part of the unit class that can be modified at will, while set-in-stone functionality like actual movement is ‘archived’ in World.

Similar thought process occurred for handling things like vision, memory, pathing and more, though quite often a change of heart occurred that required reformatting or rolling back. The worst case of this would be the status effect class’ call to try and infect a Unit, going through a loop of calling functions between the World and Status that requires said effect to be initialized and assigned to its unit target even if it ends up being blocked.

4.4. Expected game flow

The overall game flow can be characterized by a cycle of exploration, fighting and recuperation, repeated until all objectives are fulfilled, even if every enemy lies slain and no exploration is left.

At the beginning of each level, a short dramatic prologue (presented either as a dialog with nearby allies, or as a monologue if the player is alone) describes the player's situation, motivation and objectives. For example, in the introductory tutorial level, the player learns that they have assumed the role of a newly-hired commander assigned to report to a base camp located nearby. However, a recent earthquake has blocked direct access, requiring a search for miners who can be recruited to clear a path. Once the base is reached, the player learns that a neighboring nest of giant spiders have been bringing local mining operations to a halt. This affords all the excuse needed for their immediate extermination.

The player starts out with only the bare essentials in terms of units and structures, with development requiring time to mine and process the resources. In the meantime, players are expected to be scouting themselves and doing as much as they can: player characters were intentionally made unable to mine to discourage them from feeling obligated to work in that repetitive area.

Waiting in one place for every possible resource to be extracted and all upgrades maxed out, referred to as 'turtling' in the RTS community, is still possible but indirectly discouraged by the intentional lack of a 'wait for x turns' function along with many easy short-term goals manually added into each level that should be more appealing than waiting. Overall, at some point the player goes out exploring, finding enemies and obstacles as well as rewards and treasures, and at some point will come back due to injury or a lack of inventory space to collect more loot. Supplies are made to be consumable to encourage this behavior: trinkets and potions can be used only once, ammo or throwable items are easily wasted, and long-lasting weapons and armor eventually become outdated as the enemies grow stronger farther from home.

The return to the base is a time to get updated about events that may have occurred while the player was away, such as raids or new discoveries, and gives the player a chance to ponder what needs doing as they restock themselves and deposit their findings for safekeeping if necessary. Construction and micromanagement is expected to happen in bursts at first, coinciding when the player returns from adventuring, but additional methods and tools for staying in touch will reveal themselves throughout the game, allowing a constant line of communication and direction over the workforce from afar while player-led excursions are underway.

4.4.1. Expected game anti-flow

Much of the experience of the game comes from interrupting this expected flow of gameplay, highlighting the features of the fog and memory system that are unique and interesting. One of the very first things we try to show players is how memories are accurate, but can quickly become outdated: they're given a preview of the path leading straight to their base, but come 5 steps later they see rocks that weren't there before and come to realize they took the place of the previously empty ground. Enemies appear that will try and memorize patrol routes, waiting for caravans returning with a good haul before striking or picking off lone scouts if an opportunity presents itself. Spellcasters in the shadows can implant false memories into scouts that inevitably lead back to the player; things like fake dragons and demons or piles of gold and gems meant to lead them into an ambush. These mechanics are meant to get the player into a state of thinking about how reliable their information and beliefs really are at any given moment, a skill rarely exercised in the genres this game is related to.

Figures 36 and 37 (below) illustrate a typical spider strategy for picking off miners who stray too far from the safety of their base.

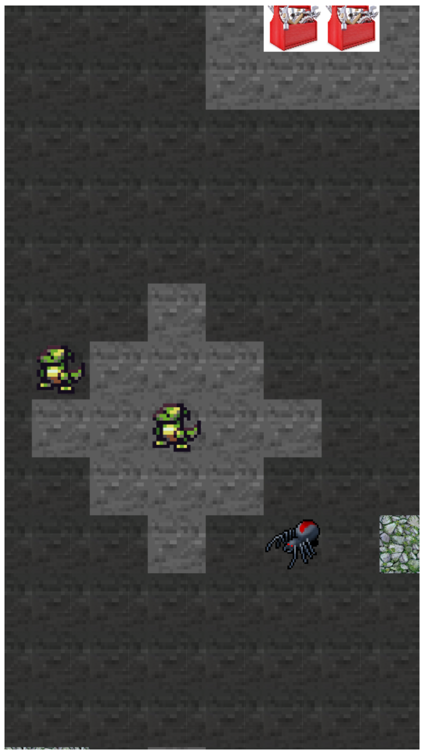


Figure 36. The Seeker from before has informed the spider base of where the miners are, and a hunting Spider sneaks up on a miner returning from a trip. Source: Screen capture.

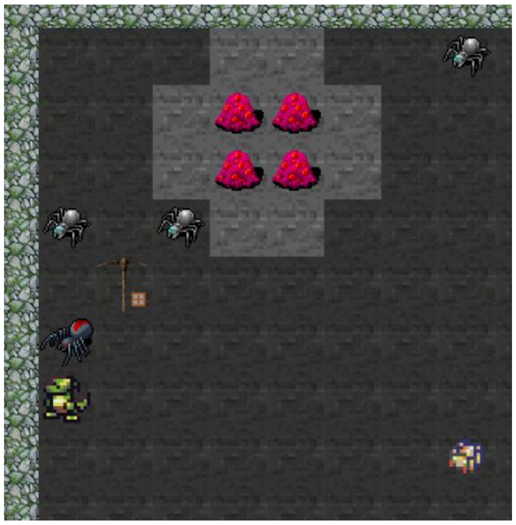


Figure 37. The spider will poison as many miners as it can until confronted by a soldier or anyone else who poses an actual threat. If undisturbed, the poison eventually numbs the victim, allowing the spider to drag it to its home nest and let the spiderlings feed and grow to become hunting spiders themselves. If not accounted for early and the nest tracked down, they become a serious threat. Source: Screen capture.

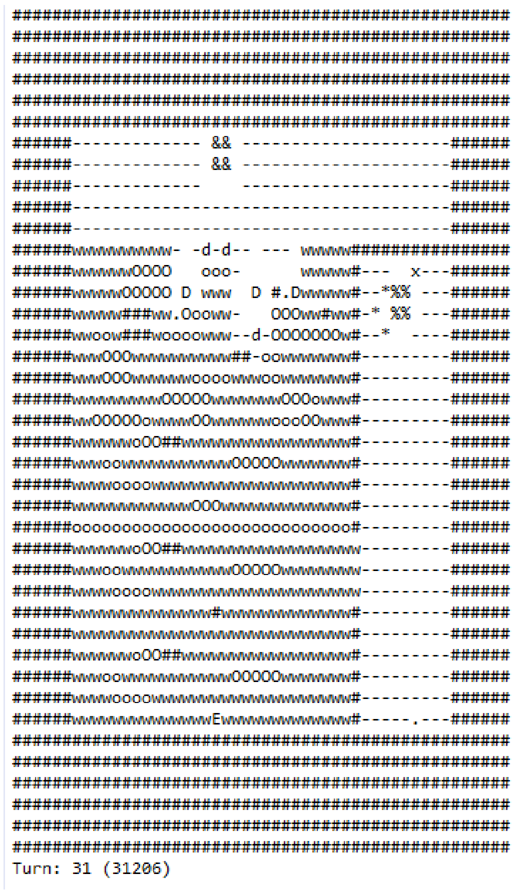
5. Graphics, sound and controls

The artistic side of game development can often be just as difficult as making the game, which turned out to be the case during *Trick of the Light*'s development. Despite the numerous intricate systems explained in other sections, getting the themes of the game expressed on the screen was a whole new challenge whose refinement was a very grueling process.

5.1. Art

Visual art considerations for this game should have been a primary concern, but being a one-man team with a focus on technical development, this aspect of development was often relegated to decisions about how to economically present necessary concepts to a player. The SDL2 library was used more for its simplicity and readily-available tutorials rather than any sort of artistic preferences, in fact being more comforting that other engines with advanced features that were sure to go underutilized. (SDL2) The features we did use were used often, with many 'cheap tricks' or roundabout ways of solving problems that would likely be handled much better by someone with more expertise in the graphical design field.

5.1.1. Roguelikes-alikes



```
##################################################
##################################################
##################################################
##################################################
##################################################
##################################################
######------------- && --------------------######
######------------- && --------------------######
######-------------    --------------------######
######-------------------------------------######
######-------------------------------------######
######wwwwwwwwwww- -d-d-- --- wwwww################
######wwwwwwOOOO   ooo-       wwwww#---  x---######
######wwwwwOOOOO D www  D #.Dwwwww#--*%% ---######
######wwwww###ww.Ooowww-   OOOww#ww#-* %% ---######
######wwoow###woooowww--d-OOOOOOOw#--*  ----######
######wwwOOOwwwwwwwwwww##-oowwwwwww#---------######
######wwwOOOwwwwwwoooowwwoowwwwwww#---------######
######wwwwwwwwwOOOOOwwwwwwwOOOowww#---------######
######wwOOOOOowwwwOOwwwwwwoooOOwww#---------######
######wwwwwwoOO##wwwwwwwwwwwwwwwwww#---------######
######wwwoowwwwwwwwwwwwOOOOOwwwwwww#---------######
######wwwooooowwwwwwwwwwwwwwwwwwwwww#---------######
######wwwwwwwwwwwwwOOOwwwwwwwwwwwwww#---------######
######oooooooooooooooooooooooooooo#---------######
######wwwwwwoOO##wwwwwwwwwwwwwwwwwww---------######
######wwwoowwwwwwwwwwwwOOOOOwwwwwwww---------######
######wwwooooowwwwwwwwwwwwwwwwwwwwwww---------######
######wwwwwwwwwwwwwwww#wwwwwwwwwwwwwww#---------######
######wwwwwwwwwwwwwwwwwwwwwwwwwwwwwww#---------######
######wwwwwwoOO##wwwwwwwwwwwwwwwwwww#---------######
######wwwoowwwwwwwwwwwwOOOOOwwwwwww#---------######
######wwwooooowwwwwwwwwwwwwwwwwwwwww#---------######
######wwwwwwwwwwwwwwwEwwwwwwwwwwwwwww#-----.---######
##################################################
##################################################
##################################################
##################################################
##################################################
##################################################
Turn: 31 (31206)
```

Figure 38. Playing *Trick of the Light* with debug mode turned on looks like this. With no rendering limit, it is possible to play up to 2000 turns per second to simulate extreme duration games if necessary. Source: Screen capture.

Trick of the Light (Figure 38) has a very good precedent for simplicity: roguelikes. Deriving from the 1980 game *Rogue* (Figure 39 below), this genre has ASCII-based roots highly engrained into its design, with the community at large still remaining reluctant to expand anywhere more mainstream than 2D graphics. (*Rogue*) Our grid based engine and ASCII roots fit directly in line with this kind of style, even if the gameplay mechanics were becoming distant as development progressed.

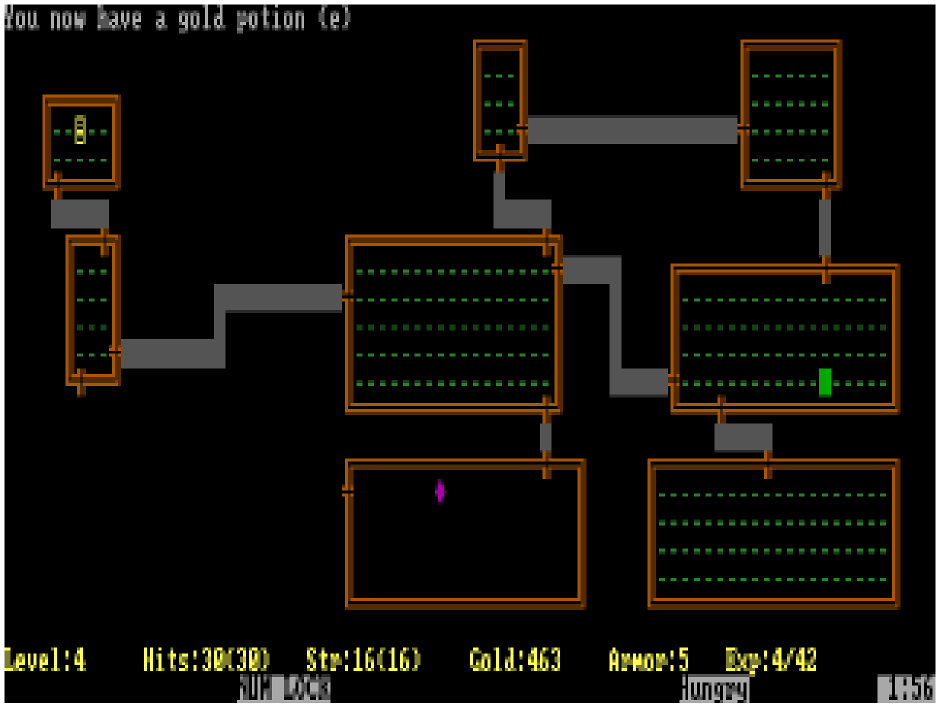

You now have a gold potion (e)
Level:4 Hits:30(30) Str:16(16) Gold:463 Armor:5 Exp:4/42
NUM LOCK
Hungry
1:56

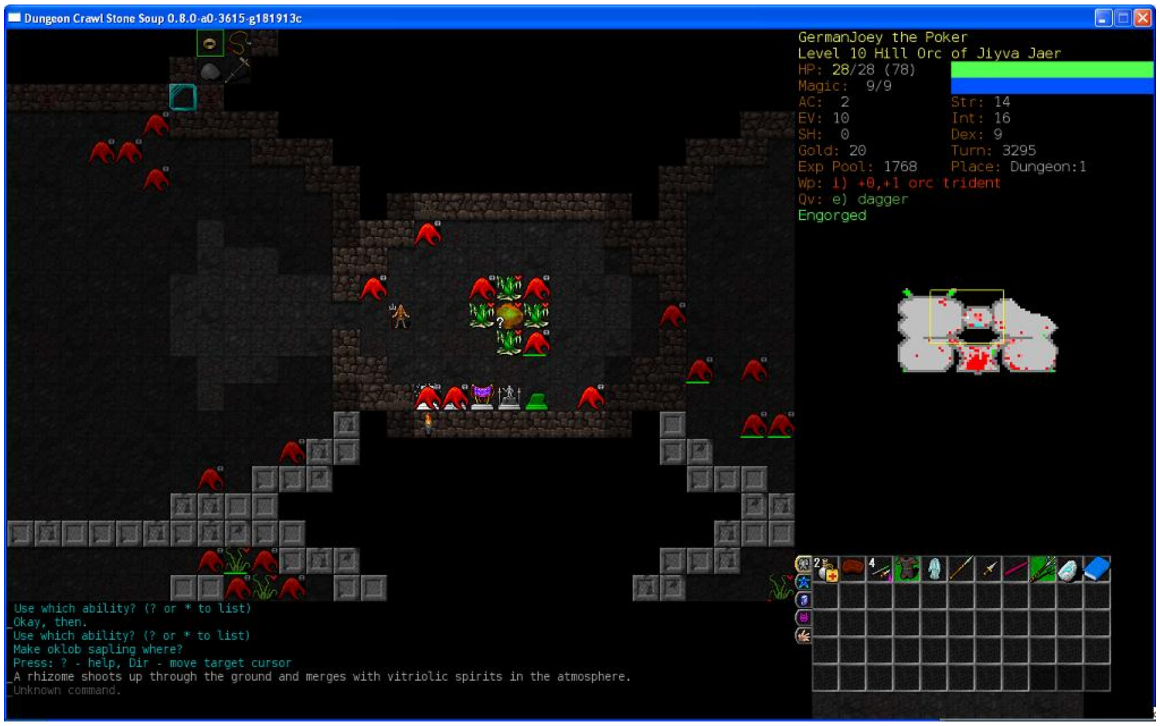
Figure 39. *Rogue* (1980), the game that defined a genre, even though it itself was based upon other ASCII adventure games and RPGS. Source: URL.

All of the art in *Trick of the Light* is tile-based, made to fit in an even 1x1 ratio within a square and fit seamlessly with its neighbors to potentially expand forever in any direction. The images depicted can sometimes mean much more than they show, or have hidden properties one can't discern from a single glance (such as what items a unit is carrying), but give the gist about what the unit is and what one can expect from it with a single still image: walls stand still, bats flap around, fighters swing swords at close range, archers run away and shoot, etc.

5.1.2. 2D Squares

Up till the end of development, the rendering process using only 2D art was extremely efficient in terms of CPU use, allowing far more than 120 FPS before a common-sense cap was put in place. The high framerate granted some extended creativity with gradual camera movement instead of instantaneous jumps, especially when zooming in and out was added in, and allowed camera controls which feel very fluid. It also unfortunately hiccupped any time a particularly complex function was being done during a turn, as a drop to 40 FPS for a split second was much more noticeable than the function that caused it in most cases. Zooming was a mixed blessing as well. This nifty feature entailed additional requirements for images which needed to be scalable and look good at any size, which doesn't coincide well with the fact most of our images were taken from free online 16x16 / 32x32 tilesets.

5.1.3. Asset acquisition



```
GermanJoey the Poker
Level 10 Hill Orc of Jiyva Jaer
HP: 28/28 (78)
Magic:  9/9
AC:  2              Str: 14
EV: 10              Int: 16
SH:  0              Dex: 9
Gold: 20            Turn: 3295
Exp Pool: 1768      Place: Dungeon:1
Wp: i) +0,+1 orc trident
Qv: e) dagger
Engorged
```

```
 Use which ability? (? or * to list)
_Okay, then.
 Use which ability? (? or * to list)
 Make oklob sapling where?
 Press: ? - help, Dir - move target cursor
_A rhizome shoots up through the ground and merges with vitriolic spirits in the atmosphere.
_Unknown command.
```

Figure 40. An example of how gameplay looks in Dungeon Crawl Stone Soup. Source: URL.

As none of the development team were great artists and the amount of individual images needed were plentiful, a savior came in the form of an online roguelike community that grants explicit permission for their assets to be used freely for any purpose, Dungeon Crawl Stone Soup. (Dungeon) Containing a multitude of available tilesets for not only units but UI and controls as well, most of the assets come directly from their extensive library (see Figure 40 above), though only as placeholders for what we would commission in the event of actual publication.

Supplementary tilesets were found on OpenGameArt.org for a few of the remaining assets, though a number of multi-square units, especially walls, lacked a perfect solution. (OpenGameArt) Some images from Game-Icons.net were also used to make the UI as consistent as possible, as the art design from Dungeon Crawl Stone Soup was a bit random at times considering how it was a community effort. (Game-Icons.net)

A complete list of art assets is provided in Appendix E.

5.1.4. Sprite-based animation

The thought of animated sprites was abandoned early on, as it would multiply the number of necessary images, but animation wasn't necessarily forgone. Rather than advance motion tweening, model / sprite warping, particle effects and whatnot, only basic SDL functionality was used, such as opacity and rotation. Combat was done simply by ramming the attacker into the defender, similar to animations done in card games like *Hearthstone* or *Magic: The Gathering*. Ranged attacks and throwing was a simple lerp from thrower to victim, sometimes with a spin or two depending on the thing being thrown. Most UI elements involved lerped movement / opacity reveals instead of flat rates or immediate transitions, such as the radial menus and vision checker, but always fast enough that an expert player who knew where things are later in the game would have to wait between clicks.

5.1.5. Fog design

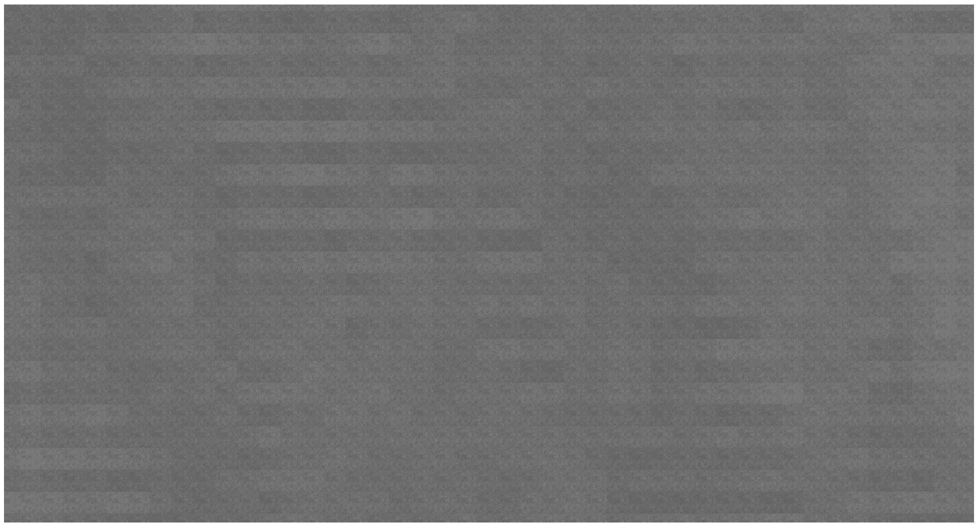


Figure 41. This still image of the rolling fog doesn't do it justice, as the 120 fps limit makes it appear much smoother and less blocky (but still blocky). Source: Screen capture.

Fog had to be the greatest artistic challenge, both for its absolute necessity as an intuitive form of vision and for the incredible variety of possible adaptations that could have been done with it. The first thing that needed to be decided with fog was whether there needed to be fog at all: though the term used to describe it is 'fog of war,' in reality we just needed a way of separating the map into what we can see now, what's been explored before, and what has yet to be explored. The 'fog' could easily just be 'darkness' that was illuminated once explored, but it was decided the effect would seem like a cop-out when the term 'fog of war' was being used so much, so some sort of fog needed to be put in.

A simple fog-image overlaying a square with less alpha than usual worked well, being able to differentiate explored vs unexplored and allowing a smooth transition simply by lerping the alpha value instead of immediately removing the fog image (see Figure 41). A cheap randomizer was the initial attempt, where the fog would appear to glide in a direction as a random fog density was passed along one square every frame, but one-directional fog was less than ideal: it appeared as if there was wind billowing the fog in one direction constantly when the theme was an underground cave. A particle engine or fluid-like techniques would have been ideal for making a swimming-ish water-like fog effect, but little was known on how to do so and early prototypes using flocking algorithms were very CPU intensive and not very appealing to look at. Manually adding permutation would provide the ripple effect we wanted, but there was a better idea.

5.1.6. Pre-generated fog

If we wanted a fog effect, we didn't have to make it generated at runtime: we could take an outside fog effect that we liked and fit it in the game. Rather than update all the variable we were using for fog every frame, we could instead convert a fog gif created in another program to a 3D array of integers: 512 x 512 to cover the span of the whole map, with 300 frames to cycle through, and each integer ranging between 0-100 depending to how heavy the fog should be in that square if applicable. Every frame the renderer would increment through to the next array, which would make the fog appear to repeat seamlessly in a way we could fabricate beforehand to be exactly what we want. A crippling problem was the amount of space the arrays took up, being almost 50mb in size. This caused crashes in the Eclipse IDE for its multi-hour indexing times. The final result was also not worth the effort, being noticeably prefabricated instead of seemingly 'natural', with the performance boost from reading an array being negligible as CPU use was the least of our concerns at the time. In addition, the fine-grain detail of the fog was actually a detriment as the size of the map tiles were much larger than the fine-grain details in the original gif, ending up looking very blocky / pixelated.

An edited version of smaller size and more fined-tuned for the size of the map was in the works, but we decided to test with the previous version of rolling-fog. In the current version, the entire map appears covered in fog with limitations on zooming and bounded movement to maintain secrecy about the real bounds of the map at first, but all the maps we have end up looking square-shaped after complete exploration, which is fine.

5.1.7. Hidden map boundaries

With the theme of exploration and the unknown being major factors of the game, we had a concern with how fog could be used to mystify the map even more. The ability to zoom in and out made a minimap unnecessary, freeing up any UI work that needed to be done in that regard but also causing the effect of having the map be unbounded by that same UI. While the map is square-shaped in the nature of its initialization, players don’t specifically know that for sure, and the lack of minimap doesn’t bind them to being at any relative location to the edge of the map. This basically means we could structure the fog so the map boundaries were never revealed, and we ended up with two different ways of accomplishing this. One was to only show fog a certain distance around explored locations, revealing more of the map from complete darkness with a very light layer of fog to show explored locations, and the other was to cover the full area of the map in fog and only allow camera movement a certain distance away from explored areas. Initial testing was done with the former version, which turned out to be very complexing for new users who couldn’t tell what the fog was representing next to the darkness and why it didn’t seem to be a complete constant around the map even when it was. When they progressed further into the map, they felt their progress was being hindered by the fog-circle surrounding them rather than more of the area being revealed as they cut through the fog, which we counted as a failure. The alternate version worked fine, but highlighted the aforementioned need for a constantly changing fog animation for the background, as a solid texture fog was unimpressive even if it felt better for exploration and unbounded the map as we desired.

5.1.8. Lighting

Lighting was put in just after fog was, unfortunately bringing light to another artistic problem. Creating light was more about creating darkness: all that changed from the previous version was an overlay of a slightly transparent dark tile above the usual one, making anything in darkness appear obfuscated while squares with light sources were untouched and much more visible. The distinction was very noticeable, especially its shape: light sources casted outwards in a circle formation, but that didn't translate well at lower distances and was very noticeably square. Incrementing one's light radius in small amounts would usually extend only a single tile in a random direction, which is less than intuitive, and at very low values wouldn't appear circular at all. Making light radiate out from the source using raycasting to check for walls would have worked, but the rendering engine would have required a complete remodel to make it work and would have been extremely difficult to make work from the player's point of view without making a few edge-case scenarios give him more information than they should know while ensuring every square was clearly recognizable as being lit or not. Figures 42 and 43 illustrate the glow radius effect onscreen.

While the ideal solution, it was pushed back for later and instead adjusted glow values were added: each glow source on a square would be checked for their distance away, with the closest source defining how bright the square was. This helped alleviate the square-ness and made the actual sources of light much more apparent, and with a bit of flickering added in it was a very convincing torch-like glow.

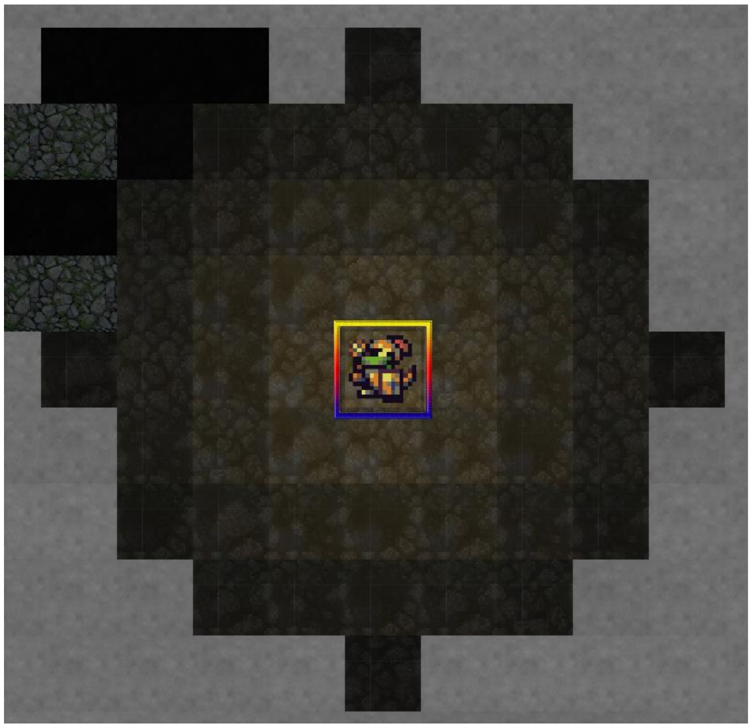


Figure 42. A glow radius is supposed to be a circle, but the result is obviously not. While the source of the light is apparent due to the gradual falloff, the ‘corners’ of the ‘circle’ are a result of a square-based rendering engine. Source: Screen capture.

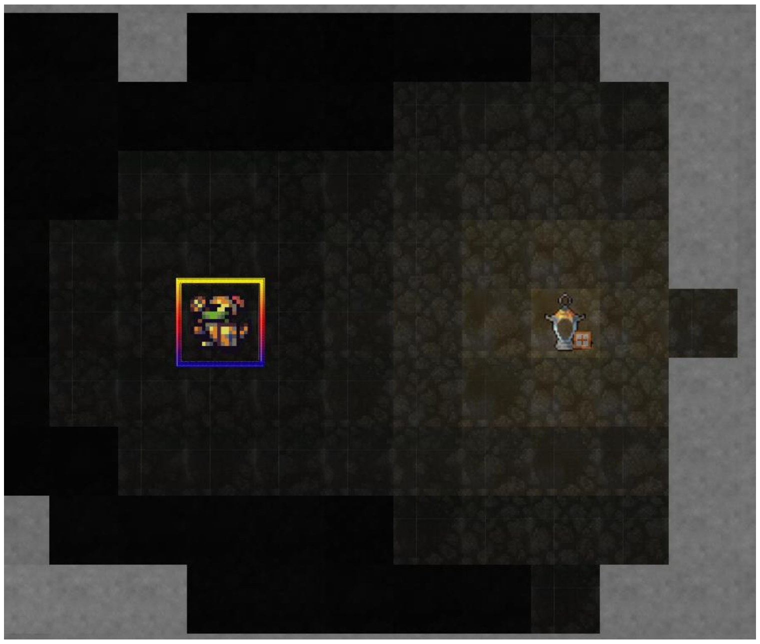


Figure 43. One of the big problems with light was finding the brightness that differentiated a lit square from a dark one, and a dark square from a dark square you could still see to because it was within your dark radius. Can you tell where the light stops and the dark radius begins? Source: Screen capture.

5.2. User interface

5.2.1. Minimalistic Style

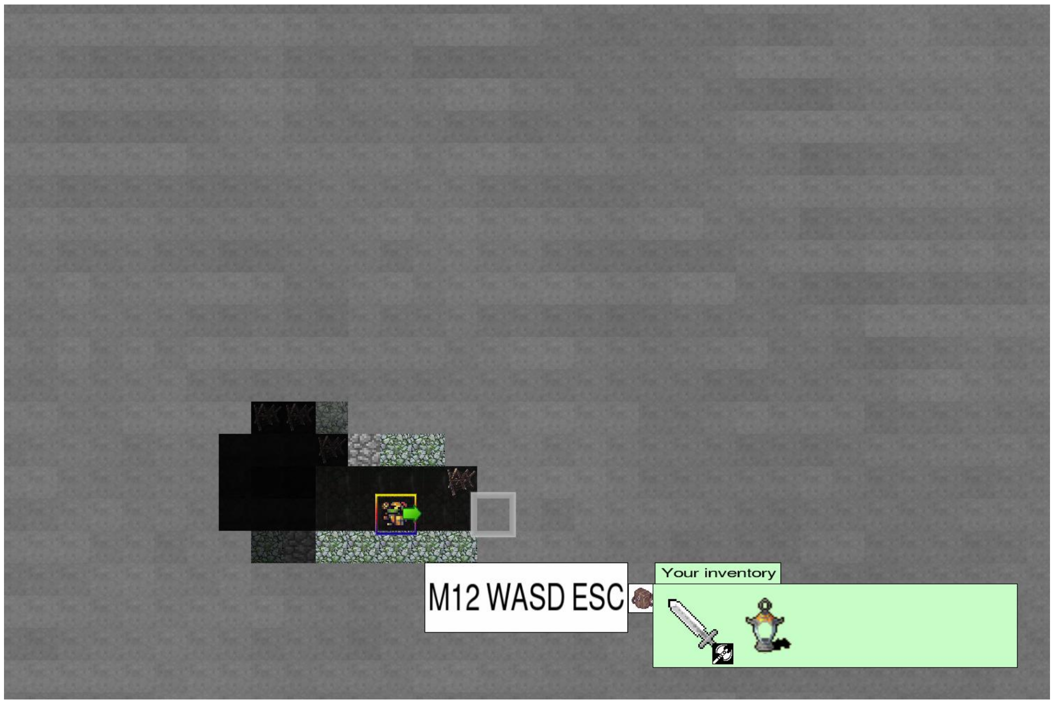


Figure 44. The average screen the player sees, with the option to minimize the bottom right inventory screen by clicking the backpack. Source: Screen capture.

The in-game UI was intentionally minimalistic, with as few elements as possible taking up constant screen space at any given moment (Figure 44). Rather than up to 25% of the lower half of the screen being reserved for controls as per a customary RTS or roguelike, only three UI elements exist: the inventory / ground / units section to the right, the help text in the center, and the ability section to the left, each with ways to minimize for maximum screen exposure. We felt no need to flood the player with all possible options from the start, and controlling their character was intended to be intuitive enough that shaking their mouse around at first would indicate how they were to interact with the world through the constantly-updating help text. The game immediately became more about looking around the screen and seeing what their character sees with no subgroups or alternate sources of attention, a much more immersive experience to assist with learning mechanics that required much more intuition than normal.

5.2.2. Menus from menus

Despite the value of minimalisim, there's are many things units can be told to do in the game, and there had to be menus to direct those actions. The inventory section doesn't take up much of the screen, but going through items brings up a menu showing what you can do with each one. If there are items or units sharing your square, you can bring up a replacement menu to decide what you do with them, which returns to the inventory menu immediately afterwards. Some actions that require a target to complete will require a second screen for choosing said target, such as picking where / who to throw an item towards or the destination of a unit being commanded to go scouting.

5.2.3. Radial menus

A radial menu scheme was devised to handle most possible interactions, including ones not normally used by AI units such as inspecting things or having a mutual trade menu. Right clicking a visible unit or item brought up a 'focus' menu from which to choose these options, following the scheme of simplistic animations by having the icons extend from the target and quickly but gradually lerp to their intended position for easy clicking. The intent was for the radial aspect to be more for quickly cycling through menus like a tree, narrowing down one's intent to a specific command from a number of available types, but we found very few testers were willing to explore much beyond the first level of menus that pops up after right clicking, and instead put more options in less menus. This in turn diluted the screen with too many options for selection, with many often not being selectable depending on whatever was being right-clicked, but more importantly confusing new players with an information overload of possible things to do.

In the end, aspects of both ideas were incorporated: the first radial menu popup was very general, showing only the option to trade, inspect, command or interact, with only the command menu leading to a variety of specific options to narrow down to.

5.2.5. Rendering loop

Rendering was handled almost totally within a single rend function, called whenever we wanted the full screen displayed in its typical tiled format with all units and items visible from someone's point of view (usually the player). Because SDL rendering overlaps everything done previously in the same bounded area, overall rendering is done in layers starting with the things we expect to possibly get replaced later.

Explored territory is drawn first, including areas the player can't see at the moment but have seen at least once, but only drawing items at visible locations. Currently visible units are drawn in the next layer, not checking the tiles themselves but rather the player's memories for units they are watching. This allows for less overlap in the case of multi-tile units that'd be drawn once for every square they were in and to lower the iteration parameters to only units we cared about at the moment.

Next, deep fog is rendered over unexplored territory, including adjusting the random values that make the fog 'roll' northwards over time, though if the array-version of fog is ever used it's a simple uncommenting of a single line to adapt. Next, out-of-sight memories are drawn, checking from what the player remembers but didn't draw in the previous section and putting their transparent silhouettes over where they think they are in the fog before finishing up and displaying the final image. In each case where units and items were being drawn, checks for overlapping occurs where additional units / items beyond the first are instead symbolized with a blue or orange plus symbol to indicate there's more things sharing that square. Right clicking these packed squares lets the player pick which one they want to focus in on.

5.2.6. Animation timers

Though the animations were acceptable, they attempted to complete two opposite tasks at once: be concise enough to allow seamless gameplay while also ensuring every important action was displayed to the player. This problem was merely mentioned in the radial section above, but the real problem was unit movement. Movement was merely sliding the Unit's sprite from one position to another, and when coming in / out of fog also giving a ripple-like effect to attract some attention. If the player's character alone is on the screen, clicking to move was reasonably fast enough to keep up with an average player's clicking, taking about 20 frames on a 120 FPS limit for the full animation and returning to wait for player input. The problem arose when the player had a group of units who were also moving: their movement animations added up, sometimes involving a number of smaller units occupying the same square moving the same direction, taking long periods of time to show each individual movement.

The first attempt to solve the problem was an animMult double that controlled the length of each animation, starting at 1.0 for the original length and reducing by 10-20% every time an animation played to hurry along the long chains. While much faster than before, it became too fast to actually detect who was being moved at high speeds: enemies could appear from the fog in front of them, or a follower may have been led astray due to some mischief and the player wouldn't notice in the increasing flurry of movement as the continuously clicked. Resetting the animMult timer back to 1.0 after each movement wasn't a good middle ground either, causing both problems at once instead of solving them.

Eventually, instead of the stream-of-consciousness way of rendering inter-turns for the player where animations only played after the last one was done, a collection of movement and attacking actions were recorded and played near-simultaneously, greatly concatenating groups of units moving or attacking at once, and keeping the aniMult properties as-is except for resetting once the player stopped rapidly clicking to move around.

5.2.7. Lack of text-logs

Somewhat ironically, what with the theme of memories being prevalent and the debug-text-output being retained for most testing versions, there are no text-log of any sort among the UI elements. Normally a staple in any roguelike-like game, the text log usually doubles as a combat-tracker, giving exact values behind what hit / didn't hit, and an adventure log, giving exposition text about the environment or dialogue and generally setting the mood where in-game images isn't enough.

The absence of this feature in *Trick of the Light* was completely intentional. The lack of a paper trail encourages players to be alert and attentive to the world they can see as it evolves, taking things to their own memory as an example of how easy things can be to forget when they're not explicitly recorded or there's no 'go-back' reset and retry button. Being as intuitive and immersive as possible was a common theme that hopefully was carried through successfully. Similar effects were also limited, such as damage numbers popping up after hitting things, and even health bars were begrudgingly put in as a bare minimum to help indicate when some creatures took more than one hit to kill.

5.3. Controls

5.3.1. From text to clicks

The control scheme started from its initial humble origins as text-based commands back when everything was ASCII; everything was uphill from there. Like a classic adventure game, all available commands were listed out to be typed and sent in one after another, leading to separate menus with more commands, just like how the radial system described previously worked. The first jump to keyboard and mouse was when SDL was put in, starting with using the numpad to move in any orthogonal direction. It turns out fewer computers than we'd hoped have a full 0-9 numpad in the format we wanted, where every key was mapped to the direction the player was moving, and as the left-hand side of the keyboard (the qwe-asd-zxc keys) didn't line up the same way movement had to be transferred over to the mouse.

Clicking initially moved one in the orthogonal direction clicked, but was almost immediately changed to fully pathing towards the square indicated instead: left click for a single step and right click to keep taking steps till the destination was reached. One could right click into unexplored territory as well, but the pathing wasn’t always intuitive: at every step the path was being recalculated, and with movement sometimes coming faster than the player could fully interpret, players would often watch semi-helplessly as their character tried to go sometimes the complete opposite direction of where they intended if the destination was unreachable (semi-helplessly, as they could click at any time to stop the auto-pathing but very rarely did so during playtesting). The solution was to simply not recalculate the path: the first route they saw was the one they took to the point something solid blocked their way, even if it ended up requiring multiple right clicks to reach their destination.

5.3.2. Keyboard

The keyboard wasn’t entirely abandoned, though it turns out during actual gameplay it often was. Instead of character movement, the keyboard was now solely for controlling the map: WASD was used to move the camera around, allowing one to change the view to out-of-sight locations and inspect memories in the fog, as well as the Q and E keys being used to zoom in and out and Z, X, C for refocusing the camera at predefined close, medium and far zoom levels. While necessary at times, the mouse could also be used for camera control by scrolling to the edge of the screen and zooming in / out from the mouse-pointer’s location using the mouse wheel, resulting in many playing the game one-handed without needing the keyboard controls for a majority of the gameplay. This wasn’t seen as an explicit problem, as the functionality was there if needed, and was mainly intended for more macro-oriented gameplay anyways, such as checking a recently updated map.

5.4. Sound and music

Sounds were put in far into development, just before testing. IMGD undergraduate Dave Allen created all sound and music assets, using a combination of assets he had created beforehand and new ones using Foley or synthesized tones. Sound effects were very short, often less than a second, and included menu-related noises like clicking or selection pips as well as in-game effects like swinging a sword or lighting a torch. “Music” was implemented as a collection of ambient noises made to sound like the area one was traveling around, looped until moving into another area caused the track to switch. However, only the default open-area track is currently used, regardless of the player’s location.

A complete list of audio assets is provided in Appendix E.

6. Testing

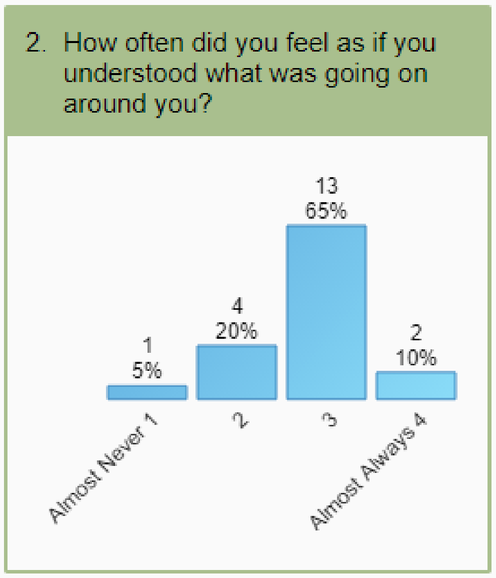
Testing was conducted using 20 IMGD undergraduate students as subjects, playing simultaneously on separate PCs in the IMDG lab. Every subject completed an IRB Informed Consent Agreement (Appendix A) before beginning.

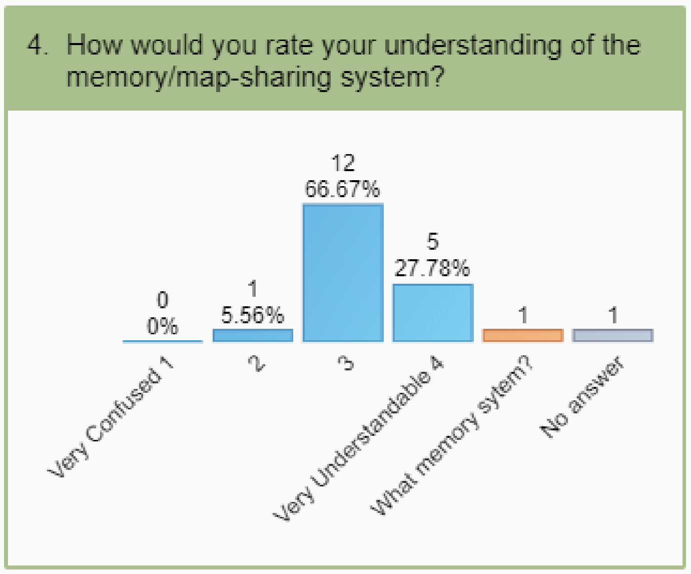
The tutorial level (illustrated in Appendix G) challenged testers with tasks involving the vision and memory-related mechanics. The goal was to see if players would understand these concepts well enough to successfully complete the tasks, using an online post-test survey to solicit their subjective opinion of the new systems.

Playtesters were encouraged to express their thoughts and ask for help anytime during the test session. Testers were not observed as they filled in the surveys to minimize any influence by the presence of the developer.

6.1. Results

The post-test survey included 1-4 Likert rankings of specific aspects of the game, as well as four questions requiring short written responses. The survey instrument is reproduced in Appendix B, with the complete results available in Appendix C.





In general, the results indicate that playtesters were generally able to understand the mechanics being presented, but experienced some trouble fully utilizing them. The theme-relevant questions, related to knowing what was going on around them in terms of vision, memories and lighting, all tested positively. "How often did you feel as if you understood what was going on around you?" had 75% reply with 'Often' or 'Almost Always,' while the question "How would you rate your understanding of the memory/map-sharing system?" received over 85% saying the system was 'Understandable' or 'Very understandable.'

However, the above results are not, by themselves, an adequate way to assess comprehension. The understanding of a concept cannot be determined simply by asking "Did you understand the concept?", especially in a setting of imperfect information in which many unknown things may be happening that a player doesn't know they're not reacting to.

In addition, the developer’s presence in the room during testing can influence the behavior of test subjects. Some may be reluctant to disappoint the developer, even if they are specifically instructed to respond as impartially and honestly as possible.

The written survey question asking testers where they stopped playing provided more impartial data. The similarity between the number of players who reported understanding the mechanics in questions 2 and 4 (averaging 80%) and the number who reported completing the tutorial (about 70%) suggests that the latter players successfully acquired the knowledge necessary to progress.

Physically being there to observe them as they played, being asked questions about said themes and listening as some spoke their thoughts out loud as they played, confirmed their understanding in ways that are more ambiguous in the written sections. Many of the questions were related to interactions not specified in-game, clarifications about the way memories are shared, or even just asking about how a new unit or interaction could be added in with the mechanics they knew about. In a few cases, questions evolved into discussions about the potential to expand on the design and the state of similar genre-related mechanics. The suggestions for interesting and relevant additions to the game implies that players understood them well enough to imagine and actually care about extra steps that might be built on those mechanics.

However, equally vocal was the dissatisfaction with the movement scheme and irregularities with controlling allies. The question “How would you rate the difficulty of managing your own units?” received a very telling 90% saying ‘Hard’ or ‘Very hard’ with the vast majority of the responses in the “What part of the game could use the most improvement?” citing the AI followers often wandering away once out of sight. The animation section mentioned before highlighted some of the solutions to problems that occurred though the iterations, but there was always something that seemed to be slowing down gameplay related to movement that always popped up after the previous problem was fixed. And allies, while always attempting to complete their tasks in a predictable manner, sometimes acted erratically from a players point of view, usually connected with being out-of-sight when moving around corners or having long narrow corridors that results in round-about pathing.

The placeholder images and minimalistic animations didn't seem to cause any backlash at all (possibly because the testers were just being courteous) and the fog and sonar-reveal effects were praised, but the UI was mentioned as a problem when trying to learn all possible actions or attempt them. There was some confusion about where to go exploring next at any given moment, as evidenced by the low average of scores in the "How often did you feel confident about where you should go/explore next?" question with a 65% 'Almost Never' or 'Not Often' answer. This was somewhat intentional, considering the efforts we made to make the map boundaries appear indistinguishable, but written responses reacting poorly indicated some more effort should be made to encouraging scouting in every direction to find interesting leads a player would jump for themselves.

7. Postmortem

7.1. What went right

The game resulting from our research and experimentation feels like it holds up under the weight of being a hybrid of so many familiar genres. The sense of adventure and intrigue that emerge from the limited vision and small-scale interactivity was an experience goal we believe we have achieved, and the RTS roots of base management and large-scale goals add strategic depth and autonomous handling of usually boring micromanagement tasks. Testers showed genuine appreciation of the game's novel mechanics and expressed interest in the project's development. Having a single technical developer implement an entire game engine from scratch provided a unique opportunity to learn about many different aspects of software architecture, and how to customize common algorithms for specific purposes.

7.2. What went wrong

The most difficult part of development was the artistic portion of the game. At the start of development, the engine was nearly complete and completely playable in an ASCII manner, but only by the designer who knew what everything represented and was able to extrapolate the systems of paranoia and limited vision from a bunch of D's and F's moving along a debug text log. Making those concepts into a sharable experience was much harder to master than any technical aspect of the project, because we didn't know what the best possible solution was for getting our thoughts onto the screen. Many of the game's features and mechanics are only felt indirectly or weren't able to be fully implemented because of this design bottleneck, resulting in only core aspects of the game being satisfactorily presented.

The testing results also indicate clear problems with the playability of the game, primarily due to the difficulty in commanding allied AI and a few flow-breaking aspects of UI and animation. While the speed issues are superficial, the problems concerning the AI behaving erratically are deeply entangled with the challenge of implementing fully independent entities with personalized memories. While it would be easy to simply make allies cheat and use the player's location more often than they should actually know, the primary purpose of the fog of war being escalated to these levels was to bring forth that level of separation on a universal scale, with anything less being a clear violation of the founding intent and a failure to deliver that world consistently.

8. Future development

Trick of the Light will continue to be worked on post-graduation, though with no immediate plans for publication. Many more iterations of testing and refinement, not to mention a complete overhaul of the game art, would need to be completed before any serious attempt to bring the game to market. Nevertheless, the game's genre-defying concepts have garnered enough interest from testers to suggest it is worth offering to the public eventually.

Its campaign-style gameplay would allow for an incremental release, delivering packs of maps filled with different challenges and races. At the very least, the game will continue to be refined solely as a point of pride, adding new features regardless of who else is interested.

9. Conclusion

Trick of the Light was a pet project that was elevated to thesis status, becoming a game about unshared fog of war and the related systems that developed from it. The concepts of individualism and propagation of information were sufficiently expanded to create a playable game, teaching numerous complex mechanics in an intuitive and immersive manner, though most of the difficulty and effort in development was presenting those concepts to the players.

Its new and potentially confusing mechanics received a positive reaction during playtesting, sparking playtester's imaginations and intrigue, and encouraging future work in development of the engine and ideas. The negative feedback involving the controlling of allied units indicates a clear need to make more player-centric design choices in future development.

Works Cited

Bergström, Björn. *FOV using recursive shadowcasting.* Roguebasin. 16 May 2017. URL: <http://www.roguebasin.com/index.php?title=FOV_using_recursive_shadowcasting>

Burgun, Keith. *Fog of War in Push the Lane (and strategy games, generally)*. 27 April 2017. Web. URL: <http://keithburgun.net/fog of war-in-push-the-lane-and-strategy-games-generally/>

Burgun, Keith. *Uncapped Look-Ahead and the Information Horizon*. 30 December 2014. Web. URL: <http://keithburgun.net/uncapped-look-ahead-and-the-information-horizon/>

Dungeon Crawl Stone Soup. Web. 3 February 2018. URL: <https://crawl.develz.org/>

Game-Icons. Web. July 2017. URL: <http://game-icons.net/>

Guillory, Brant. *What don't we know about what we don't know that we don't know?* Grogheads. Web. 15 July 2015. URL: <http://grogheads.com/featured-posts/8596>

Kiesling, Eugenia. *On War without the Fog.* Military Review. Web. September 2001. URL: <https://www.clausewitz.com/bibl/Kiesling-OnFog.pdf>

Lewin, Christopher George. *War Games and their History*. Fonthill Media, 2012. Print. ISBN 978-1-78155-042-7.

Milazzo, Adam. *Roguelike Vision Algorithms*. Web. 8 October 2014. URL: <http://www.adammil.net/blog/v125_roguelike_vision_algorithms.html>

Open Game Art. Web. 15 January 2018. URL: <https://opengameart.org/>

Register, Steve. *Simple Line of Sight*. Roguebasin. 15 December 2014. URL: <http://www.roguebasin.com/index.php?title=Simple_Line_of_Sight>

Rogue. A.I. Design. 1980. Video game.

SDL (Simple DirectMedia Layer). Open-source cross-platform development library. URL: <http://www.libsdl.org/>

Setear, John. *Simulating the Fog of War*. The RAND corporation. Web. February 1989. URL: <http://www.dtic.mil/dtic/tr/fulltext/u2/a228112.pdf>

SimAnt. Maxis. 1981. Video game.

Tiled. Web. 2017. URL: <https://www.mapeditor.org/>

Vandevenne, Lode. *Lode's Computer Graphics Tutorial*. Web. 2018. URL: <http://lodev.org/cgtutor/raycasting.html>

Warcraft 3. Blizzard. 3 July 2002. Video game.

Wayward. *Battlefield Uncertainty and Fog of War. Wayward Strategist*, 30 January 2015. Web. URL: <https://waywardstrategist.com/2015/01/30/battlefield-uncertainty-and-fog of war/>

Appendix A: IRB Informed Consent Agreement

Informed Consent Agreement for Participation in a WPI Research Study

Investigator: Brian Moriarty, IMGD Professor of Practice

Contact Information:

Brian Moriarty

bmoriarty@wpi.edu, 508 831-5638

Title of Research Study: Unshared Fog-of-War Experiment

Sponsor: WPI

Introduction: You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: The purpose of this study is to obtain playtest feedback in order to locate/address operational bugs, to identify opportunities for design improvement, and to gather data to conduct statistical analyses on to measure games effectiveness towards the experience goal.

Procedures to be followed: You will be asked to play a brief game lasting less than thirty minutes. After completing the game, you will be asked to complete brief, anonymous survey describing your subjective experience. Any responses you offer will not be associated with your name or any other personally identifiable information about you.

Risks to study participants: There are no foreseeable risks associated with this research study.

Benefits to research participants and others: You will have an opportunity to enjoy and comment on a new game under active development. Your feedback will help improve the game experience for future players.

Record keeping and confidentiality: Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to this confidential data. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: There is no foreseeable risk of injury associated with this research study. Nevertheless, you do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact the Investigator listed at the top of this form. You may also contact the IRB Chair (Professor Kent Rissmiller, Tel. 508-831-5019, Email: kjr@wpi.edu) and the University Compliance Officer (Jon Bartelson, Tel. 508-831-5725, Email: jonb@wpi.edu).

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

______________________________________ Date: ____________________

Study participant signature

____

Study participant name (please print)

______________________________________ Date: ____________________

Appendix B: IRB Study Purpose and Protocol

In addition to the playtesting survey, our intention was to poll several prominent experts on game design (Sid Meier, George Phillies and Chris Crawford) on a single question regarding their experiences with fog of war:

What map-based analog or digital games have you encountered that employ particularly effective, creative and/or unusual implementations of (1) fog of war and/or (2) the propagation/transfer of knowledge about the current map state?

It was hoped that their responses to this question would contribute to the development of *Trick of the Light's* mechanics. Unfortunately, the IRB protocol describing the proposed queries was approved very late into development. The emails were sent out regardless, but no replies were received in time for inclusion in this report.

Title: Unshared Fog-of-War Experiment

1. Purposes of study

a. To obtain playtest feedback in order to locate/address operational bugs in the game, and to identify opportunities for design improvement.

b. To solicit the opinion of domain experts regarding the most effective, creative and/or unusual implementations of fog of war and map-state knowledge propagation/transfer they have encountered among analog and digital games.

2. Study protocol for playtest feedback

Participants are provided a computer on which to play the game. Investigators observe participants during play. Afterward, participants are asked to fill out a short survey to characterize their subjective experience.

2.1. Opening briefing for playtesters

“Hello, and thank you for volunteering to test my game. Before we begin, could you please read and sign this Informed Consent form?”

[Subject signs Informed Consent form.]

“Thank you. When your play session is complete, I will ask you to complete a brief survey about your play experience. At no point during your play session, or in the survey after, will any sort of personal and/or identifying information about you be recorded. Please begin playing when you feel ready.”

2.2. Post-Playtest Survey Questions

[Note: Space will be provided for optional comments after each question.]

All questions are optional. Respond to as few or as many as you want.

1. How would you rate the effectiveness of the tutorials in explaining how to play?

1-4 Likert scale, 1 = Poor, 4 = Excellent

2. How often did you feel lost or uncertain about your location while exploring?

1-4 Likert scale, 1 = Almost never, 4 = Nearly always

3. How often did you feel as if you understood what was going on around you?

1-4 Likert scale, 1 = Almost never, 4 = Nearly always

4. How often did you feel confident about where you should go/explore next?

1-4 Likert scale, 1= Almost never, 4 = Nearly always

5. How would you rate your understanding of the memory-sharing system?

1-4 Likert scale, 1 = Poor, 4 = Excellent

6. How often did you feel dependent on the vision-sharing system in order to progress?

1-4 Likert scale, 1= Almost never, 4 = Nearly always

8. How would you rate the difficulty of managing your own units?

1-4 Likert scale, 1 = Difficult, 4 = Easy

9. How would you rate the overall difficulty of the game?

1-4 Likert scale, 1 = Difficult, 4 = Easy

10. Did any aspects of the game seem particularly unusual or unexpected?

Blank field for written response

11. Do you have any general comments/feedback regarding your game experience?

Blank field for written response

3. Study protocol for solicitation of expert opinion

Three publicly-known professional game designers (Sid Meier, Chris Crawford and George Phillies, all personal acquaintances of the principal investigator) will be contacted via email, explained the purpose of the thesis and invited to voluntarily respond to the following question:

What map-based analog or digital games have you encountered that employ particularly effective, creative and/or unusual implementations of (1) fog of war and/or (2) the propagation/transfer of knowledge about the current map state?

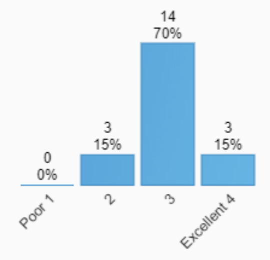
Key quotations from consenting respondents will be incorporated into the body of the thesis report. Complete transcripts of all responses will be included as appendices in the report. Respondents will be given an opportunity to review and approve the response text attributed to them before report publication.

4. Hazardous materials/special diets

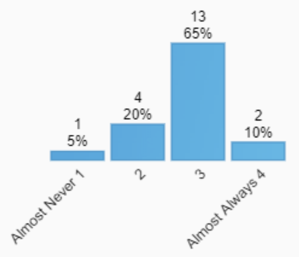
No hazardous materials or special diets are involved in this study.

Appendix C: Post-test survey results

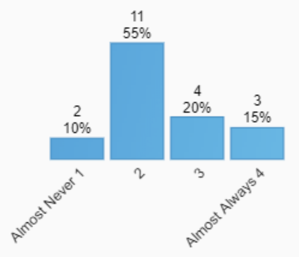
1. How would you rate the effectiveness of the tutorials in explaining how to play?



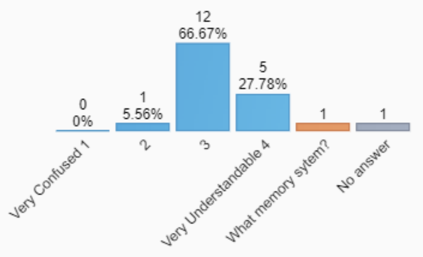
2. How often did you feel as if you understood what was going on around you?



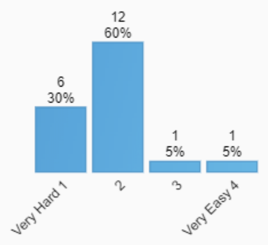
3. How often did you feel confident about where you should go/explore next?



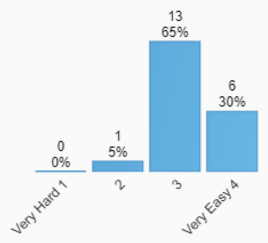
4. How would you rate your understanding of the memory/map-sharing system?



5. How would you rate the difficulty of managing your own units?



6. How would you rate the overall difficulty of the game?



Appendix D: Post-test survey data

(20) 2018-03-20 15:32:31	×
1. How would you rate the effectiveness of the tutorials in explaining how to play?	4/4
2. How often did you feel as if you understood what was going on around you?	2/4
3. How often did you feel confident about where you should go/explore next?	2/4
4. How would you rate your understanding of the memory/map-sharing system?	3/4
5. How would you rate the difficulty of managing your own units?	2/4
6. How would you rate the overall difficulty of the game?	3/4
7. At what point in the game did you stop playing and why?	I stopped playing after I got miners to help me mine ores but could not equip my own pick axes. They were crossed out in my menu even when I didn't have another weapon equipped. I felt like it would be too difficult if I could not mine on my own.
8. Did any aspect of the game seem particularly unusual, interesting or unexpected?	The walking mechanic is hard to get used to, probably because is it relatively slow. I don't think the player character should have to be within one block of another object to interact with it--2 blocks away seems more natural.
9. What part of the game could use the most improvement?	Playing speed - I am trying to play faster than the game allows; it feels limiting.
10. Do you have any general comments/feedback regarding your game experience?	I really like the exploration aspect and figuring out the mechanics!

(19) 2018-03-20 15:28:47	×
1. How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2. How often did you feel as if you understood what was going on around you?	3/4
3. How often did you feel confident about where you should go/explore next?	2/4
4. How would you rate your understanding of the memory/map-sharing system?	3/4
5. How would you rate the difficulty of managing your own units?	2/4
6. How would you rate the overall difficulty of the game?	2/4
7. At what point in the game did you stop playing and why?	I stopped playing after I filled out the entire map.
8. Did any aspect of the game seem particularly unusual, interesting or unexpected?	Not really.
9. What part of the game could use the most improvement?	The ally management system. It was confusing to get map info, and when I tell them to follow me, the units would most of the time go off on their own.
10. Do you have any general comments/feedback regarding your game experience?	This is a good game idea, just not my cup of tea.

(18) 2018-03-20 15:27:22	×
1. How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2. How often did you feel as if you understood what was going on around you?	2/4
3. How often did you feel confident about where you should go/explore next?	4/4
4. How would you rate your understanding of the memory/map-sharing system?	3/4
5. How would you rate the difficulty of managing your own units?	2/4
6. How would you rate the overall difficulty of the game?	3/4

7.	At what point in the game did you stop playing and why?	When I got to the buildings, and found out that you couldn't yet interact with them
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	It was weird that it kind of suddenly turned from an adventure game into an RTS
9.	What part of the game could use the most improvement?	INVENTORIES. it's inconvenient and frustrating that in the initial exploring part of the game, you didn't have enough inventory space to carry everything you found. Additionally, the "escape" information panel doesn't contain information on dropping/equipping items, so I need to stumble around until I figured it out.
10.	Do you have any general comments/feedback regarding your game experience?	If this is supposed to be an RTS-style game, then I feel that the beginning adventure phase is a little too long.
(17) 2018-03-20 15:20:10		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	3/4
4.	How would you rate your understanding of the memory/map-sharing system?	4/4
5.	How would you rate the difficulty of managing your own units?	1/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	Once most of the areas had been explored, leaving only a few hidden behind mine-able rock. It was too frustrating to try and lead

		miners to those areas to mine the rock for me.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	Light and map sharing system was very interesting. Every time I shared a map with an ally it felt like an accomplishment. Seeing the map revealed was rewarding.
9.	What part of the game could use the most improvement?	AI pathfinding abilities. Your followers too easily get lost or distracted.
10.	Do you have any general comments/feedback regarding your game experience?	Very interesting concepts, enjoyable to play. If AI can be sharpened up or the player didn't have to rely on them as much, it might improve things.
(16) 2018-03-20 15:14:11		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	3/4
4.	How would you rate your understanding of the memory/map-sharing system?	
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	I stopped playing once I accidentally got off the map pressing the map moving keys. I couldn't access my characters at that point anymore, and when I got back onto a map everything was set up at different places but I couldn't move any characters. I think I probably played it wrong, but I don't

		exactly know what happened there.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	I liked the right-click commands.
9.	What part of the game could use the most improvement?	Probably the icons and most especially the movement of the units under your control. They move very randomly after the player character moves, and if you're not careful you'll have to go back and forth just to have your party in one place. Also, add a function to make the player move more than one tile, or just shorten the map. It gets tedious clicking once and then waiting for them to make their one tile move
10.	Do you have any general comments/feedback regarding your game experience?	It was slow, but it was rewarding.
(15) 2018-03-20 15:11:39		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	3/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	End of tutorial, then explored the remainder of the navigable map. No further content.
8.	Did any aspect of the game seem particularly unusual,	The NPC's wandering by

interesting or unexpected?	were rather strange, it was a bit annoying to chase them down to interact with them because some of them seemed to be doing their own thing and would not respond to follow requests.
9. What part of the game could use the most improvement?	2 major changes: - The freezing on the map sharing mechanic really breaks the flow of the gameplay - NPC's just disappear when going off screen and don't always keep up with the player
10. Do you have any general comments/feedback regarding your game experience?	- Making intro text sequence more concise, explaining mechanics a bit more visually as opposed to just paragraphs would make it much smoother - Minor detail, but it would be nice to be able to use the keyboard for navigation more (especially using space to go through tooltips instead of having to click)
(14) 2018-03-20 15:10:13	×
1. How would you rate the effectiveness of the tutorials in explaining how to play?	4/4
2. How often did you feel as if you understood what was going on around you?	2/4
3. How often did you feel confident about where you should go/explore next?	2/4
4. How would you rate your understanding of the memory/map-sharing system?	3/4
5. How would you rate the difficulty of managing your own units?	2/4
6. How would you rate the overall difficulty of the game?	4/4
7. At what point in the game did you stop playing and	I stopped playing once it

	why?	seemed like I explored the entire cave and killed the spider base.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	I liked the concept of commanding your units around and how they worked behind the scenes whether you were seeing them work or not. It gave the game world a organic feeling.
9.	What part of the game could use the most improvement?	I felt that the GUI for commanding allies and trading items was clunky. A control scheme similar to Warcraft 3 might be more efficient and visually appealing. I also feel that the cave might benefit from a procedural generated randomness.
10.	Do you have any general comments/feedback regarding your game experience?	Interesting Concept that seems to work well. If the cave's size is extended or if new areas are able to be unlocked, the exploration and resource management will be a fun experience.
(13) 2018-03-20 15:10:11		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	3/4
4.	How would you rate your understanding of the memory/map-sharing system?	4/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and	Once I finished exploring the

	why?	map
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	I was very surprised when I realized diggers were harvesting and delivering on their own.
9.	What part of the game could use the most improvement?	Pathfinding and UI navigation.
10.	Do you have any general comments/feedback regarding your game experience?	Very neat.
(12) 2018-03-20 15:07:08		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	4/4
4.	How would you rate your understanding of the memory/map-sharing system?	What memory sytem?
5.	How would you rate the difficulty of managing your own units?	1/4
6.	How would you rate the overall difficulty of the game?	4/4
7.	At what point in the game did you stop playing and why?	I stopped playing after the tutorial had finished and I felt like I had explored most of the map. I stopped after exploring most of the map because I felt that I had seen everything the game had to offer at that point.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	The actions of the supplementary characters (the miners/diggers especially). Their movement patterns were very erratic, and trying to get them to stay with me or perform certain actions (especially mining) proved to be quite the challenge.

9. What part of the game could use the most improvement?	I think either the flow of the game or the AI need to be improved upon, more so the AI. I had a lot of trouble trying to keep my party together, even after giving them commands to follow me. They would get lost in the fog, and sometimes I would not find them until 15 minutes later in some random part of the map.
10. Do you have any general comments/feedback regarding your game experience?	I had fun playing this game. I do not play many RTS games to begin with, so waiting for each of my party members/ enemies to take their turns before I could move to another space/ perform another action was a little tedious. However, I think the exploration with the fog elements is really well integrated, and I enjoyed discovering new areas within the game. The combat could be better, as it was really easy to take down enemies and provided almost no challenge. The biggest issue is managing your AI companions, as they easily get lost behind you when you move far away from them, and they can drag you down if they are trying to mine the same block when you want them to move, for instance. Overall, the aesthetics were done well, and exploration was very fun, but the combat and map traversal could definitely be worked on.
(11) 2018-03-20 15:06:33	×

1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	3/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	I tried to interact with a block of allies and the game crashed
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	I liked the sharing system, but it was a little bit difficult to understand
9.	What part of the game could use the most improvement?	Sometimes the following commands didn't seem to be working. The tutorial could use some proofreading You missed a few apostrophes and instead of "..." ",,," appeared multiple times.
10.	Do you have any general comments/feedback regarding your game experience?	I clicked during a map exchange and the game froze for a bit. If you could show which items had been given to your allies through smaller icons, that would be helpful. In the tutorial, you instruct the player to right click to command a group of allies. This blurb appeared before I was in range to do this. I'd suggest having it show up while the player is in range. In the tutorial, sometimes dialogue boxes would appear on the right side of the screen and be

	partially cut off. I could not read all of the text because of this. I would suggest editing the tutorial text to make sure the grammar and capitalization are correct. I also didn't know what an RTS was, but the tutorial assumes that the player has this knowledge.
(10) 2018-03-20 15:05:10	×
1. How would you rate the effectiveness of the tutorials in explaining how to play?	4/4
2. How often did you feel as if you understood what was going on around you?	4/4
3. How often did you feel confident about where you should go/explore next?	3/4
4. How would you rate your understanding of the memory/map-sharing system?	3/4
5. How would you rate the difficulty of managing your own units?	1/4
6. How would you rate the overall difficulty of the game?	3/4
7. At what point in the game did you stop playing and why?	Kept going after the tutorial ended for a little bit to mine out a corner of the map, but when it led to a dead end I was sad and quit.
8. Did any aspect of the game seem particularly unusual, interesting or unexpected?	I thought the mechanic of being able to share maps with allied units was interesting. The same thing was sort of implemented in some versions of Civilization (Civ III, I think) where you could trade maps with other civs once you both learned the writing and cartography technologies.
9. What part of the game could use the most improvement?	The UI

10.	Do you have any general comments/feedback regarding your game experience?	Good concept, with some polish it could be great :)
(9) 2018-03-20 15:00:47		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	3/4
5.	How would you rate the difficulty of managing your own units?	1/4
6.	How would you rate the overall difficulty of the game?	4/4
7.	At what point in the game did you stop playing and why?	Just after where the game said the survey was done, since I was curious. The last thing I did was try to interact with the thing just south of me at that point, where I gave it the ores I had and its tiles quickly alternated between two sprites, I think.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	Units following me stopped following me surprisingly often. Led to one of the two initial allies dying when it went alone after I and the other initial ally went through a thin corridor that led to a dead end. Maybe the other ally was blocking it's vision of me or something? I dunno.
9.	What part of the game could use the most improvement?	tough question...I guess the ways you can control your allies? The miners mined at whatever was minable we came a fair distance of, and I wish there was a command

		for them to ONLY follow me.
10.	Do you have any general comments/feedback regarding your game experience?	Not really.
(8) 2018-03-20 15:00:32		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	3/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	When I reached the end of the tutorial there wasn't that much more to do than just walking around.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	You can't stray too far from miners when you go light speed because they get lost in the dark or mine something in there path rather than move around it which was kind of annoying.
9.	What part of the game could use the most improvement?	The run time for map sharing takes a few minutes to load rather than just a few seconds. Fixing this would make the experience slightly better.
10.	Do you have any general comments/feedback regarding your game experience?	Make the miners faster!
(7) 2018-03-20 14:59:26		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4

2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	2/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	4/4
7.	At what point in the game did you stop playing and why?	After I got to the castle at the end of the tutorial, it seemed to crash. It eventually recovered but at that point I didn't really know what was going on.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	The memory system seems cool, but after the tutorial I still don't quite understand it. Will the units ever learn new information if they're just following you?
9.	What part of the game could use the most improvement?	The UI often seemed unintuitive. The movement was difficult, requiring a click each turn. Why can't I just move with arrow keys? I'm not sure that two separate move commands are necessary (just have one that does the path). Holding down the movement button to keep moving would be good too. In terms of inventory management, it would be good to automatically equip new items if they are better than what is already in the slot. The page system also doesn't seem to work very well, as you can only scroll one way. It would be better to click on

	the page tabs themselves, or have a full inventory screen plus a hotbar. When dealing with allies, I wasn't sure how to get them to use the items I gave them. They should automatically equip the best weapon in their inventory. Finally, a major problem with the turn based gameplay is that the player can't move while allies are attacking inanimate objects. Could you do the same thing as Civ and make everybody take their turn at the same time unless they are in combat?
10. Do you have any general comments/feedback regarding your game experience?	In general, the UI felt like it could be made simpler. The over-reliance on multiple menus is super common in this type of game and makes them difficult to learn and adds features that many people will likely never use.
(6) 2018-03-20 14:59:11	×
1. How would you rate the effectiveness of the tutorials in explaining how to play?	2/4
2. How often did you feel as if you understood what was going on around you?	2/4
3. How often did you feel confident about where you should go/explore next?	4/4
4. How would you rate your understanding of the memory/map-sharing system?	3/4
5. How would you rate the difficulty of managing your own units?	3/4
6. How would you rate the overall difficulty of the game?	3/4
7. At what point in the game did you stop playing and why?	When the scripted tutorial stopped, and I got to the miners.

8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	- Allies disappearing when performing pathfinding to a blocked location
9.	What part of the game could use the most improvement?	- Tutorials, needs graphics for demonstration - More unified UI - The plus sign in the inventory UI is confusing; does it add more items or does it go to the next inventory page?
10.	Do you have any general comments/feedback regarding your game experience?	
(5) 2018-03-20 14:55:47		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	4/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	Shortly after the tutorial. I took a little time to explore further, but at that point, there were no more goals to complete.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	I liked that my little miner friends have a mind of their own as I traveled back through the cave. I liked that they went off to go mine a rock and then continued to follow me. At first I was like, "Wait. Friend. Where are you going?" Then It made sense.

9.	What part of the game could use the most improvement?	The UI. I'm not sure if it was the placeholder assets or the structure of the UI, but it felt very cumbersome. Especially the backpack. It was a little annoying to only see four or five items at a time and to have my inventory be in the way of portions of the map I was trying to see. I do like the right click character menu, though. I'm sure once there are uniform art assets it will feel a little better.
10.	Do you have any general comments/feedback regarding your game experience?	Overall, I think that it could be a really fun game. It's pretty buggy, which I'm sure that you're aware of. It also suffers from confusing placeholder art assets. One suggestion I have is to maybe increase the movement or turn speed? Right now traveling long distances feels a little slow and awkward. I look forward to seeing how the game evolves!
(4) 2018-03-20 14:55:23		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	2/4
2.	How often did you feel as if you understood what was going on around you?	1/4
3.	How often did you feel confident about where you should go/explore next?	1/4
4.	How would you rate your understanding of the memory/map-sharing system?	3/4
5.	How would you rate the difficulty of managing your own units?	1/4
6.	How would you rate the overall difficulty of the game?	3/4

7.	At what point in the game did you stop playing and why?	Once I freed the units, the rest of the game seemed pointless.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	Other than the AI constantly breaking when it tries to follow me moving faster than 1 square at a time, the load times were very off.
9.	What part of the game could use the most improvement?	The load times and the combat need to be improved. Combat feels boring and uninteresting.
10.	Do you have any general comments/feedback regarding your game experience?	To make the combat feels better, I guess actually commanding units to attack rather than let them auto hit.
(3) 2018-03-20 14:54:19		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	3/4
5.	How would you rate the difficulty of managing your own units?	2/4
6.	How would you rate the overall difficulty of the game?	3/4
7.	At what point in the game did you stop playing and why?	I stopped playing after the tutorial, because I believe that I had achieved the understanding needed.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	The base was not fleshed out in the tutorial, so it was confusing.
9.	What part of the game could use the most improvement?	Movement, it can get tedious when a lot of units are on the screen.

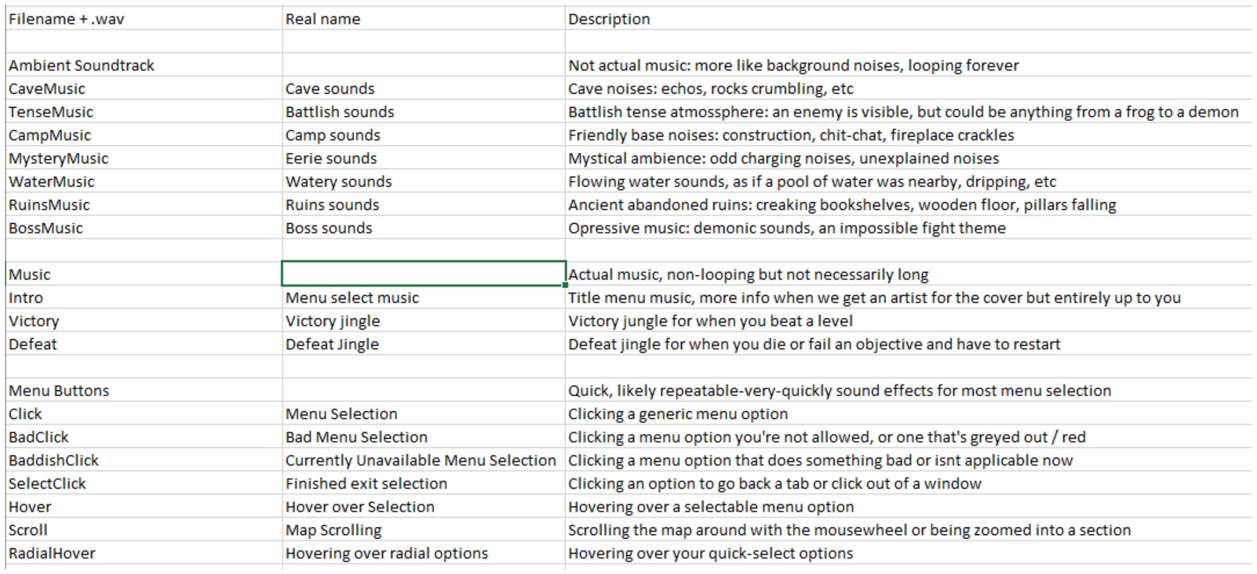
10.	Do you have any general comments/feedback regarding your game experience?	Nothing to serious when wrong for mine, but pathfinding could be improved.
(2) 2018-03-20 14:52:27		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	3/4
2.	How often did you feel as if you understood what was going on around you?	3/4
3.	How often did you feel confident about where you should go/explore next?	1/4
4.	How would you rate your understanding of the memory/map-sharing system?	4/4
5.	How would you rate the difficulty of managing your own units?	1/4
6.	How would you rate the overall difficulty of the game?	4/4
7.	At what point in the game did you stop playing and why?	After I unlocked the whole map through memories and returned to the mining camp. That felt complete to me.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	I only had AI that I got to follow me for the whole game.
9.	What part of the game could use the most improvement?	AI pathfinding. Bug fixing. Combat.
10.	Do you have any general comments/feedback regarding your game experience?	I had a lot of glitches, like I could hit enemies from anywhere on the screen, and I couldn't destroy the bat birdhouse thing no matter how many times I hit it. Also my AI and I stopped taking damage after the troll hit us both.
(1) 2018-03-20 14:50:48		×
1.	How would you rate the effectiveness of the tutorials in explaining how to play?	2/4
2.	How often did you feel as if you understood what was	4/4

	going on around you?	
3.	How often did you feel confident about where you should go/explore next?	2/4
4.	How would you rate your understanding of the memory/map-sharing system?	4/4
5.	How would you rate the difficulty of managing your own units?	4/4
6.	How would you rate the overall difficulty of the game?	4/4
7.	At what point in the game did you stop playing and why?	Stop Playing once I reached a very populated area as the turn based aspect of the game had me waiting for long intervals to move just one step.
8.	Did any aspect of the game seem particularly unusual, interesting or unexpected?	Map sharing cause large load times, Speed traveling caused companions who are set to follow to be lost, High populated areas with nothing going on causes travel to be painfully long.
9.	What part of the game could use the most improvement?	Turn based aspect, if you are in a room with people who are set to friendly or neutral status I should not have to wait for a "turn" to move as I am passing through.
10.	Do you have any general comments/feedback regarding your game experience?	I like this game conceptually.

Appendix E: Art and audio assets

Note: These lists include all assets planned for inclusion in the first full release (totalling seven levels) of the game. Only a subset was actually produced for the tutorial level completed for the submitted project.

E1. Audio assets



Filename + .wav	Real name	Description
Ambient Soundtrack		Not actual music: more like background noises, looping forever
CaveMusic	Cave sounds	Cave noises: echos, rocks crumbling, etc
TenseMusic	Battlish sounds	Battlish tense atmossphere: an enemy is visible, but could be anything from a frog to a demon
CampMusic	Camp sounds	Friendly base noises: construction, chit-chat, fireplace crackles
MysteryMusic	Eerie sounds	Mystical ambience: odd charging noises, unexplained noises
WaterMusic	Watery sounds	Flowing water sounds, as if a pool of water was nearby, dripping, etc
RuinsMusic	Ruins sounds	Ancient abandoned ruins: creaking bookshelves, wooden floor, pillars falling
BossMusic	Boss sounds	Opressive music: demonic sounds, an impossible fight theme
Music		Actual music, non-looping but not necessarily long
Intro	Menu select music	Title menu music, more info when we get an artist for the cover but entirely up to you
Victory	Victory jingle	Victory jungle for when you beat a level
Defeat	Defeat Jingle	Defeat jingle for when you die or fail an objective and have to restart
Menu Buttons		Quick, likely repeatable-very-quickly sound effects for most menu selection
Click	Menu Selection	Clicking a generic menu option
BadClick	Bad Menu Selection	Clicking a menu option you're not allowed, or one that's greyed out / red
BaddishClick	Currently Unavailable Menu Selection	Clicking a menu option that does something bad or isnt applicable now
SelectClick	Finished exit selection	Clicking an option to go back a tab or click out of a window
Hover	Hover over Selection	Hovering over a selectable menu option
Scroll	Map Scrolling	Scrolling the map around with the mousewheel or being zoomed into a section
RadialHover	Hovering over radial options	Hovering over your quick-select options

Sound Effects		In-game effects relevant to whatever characters are doing at the time
Windup	Attack windup	An attack of any type being wound up, followed by one of the effects below
BluntHit	Blunt Attack	Hitting someone with a blunt weapon, ie club
SharpHit	Sharp Attack	Slashing someone with a sharp weapon, ie sword
StabHit	Stab Attack	Stabbing someone with a sharp weapon, ie dagger
ArrowHit	Bow attack	Hitting someone with an arrow launched from a bow
BiteHit	Bite attack	Biting someone, most likely from an animal like a spider / dog
PickHit	Mining Pick Attack	Hitting something with a pick, likely a rock
Missed	Missed swoosh	Missing someone after trying to attack them
Blocked	Blocked a hit	Blocking an attack
HumanDie	Human dieing	A human character just died
AnimalDie	Animal Dieing	An animal character just died, most likely a spider
GoblinDie	Goblin Dieing	A goblin character just died
WallDie	Demolished Wall	A wall was just completely mined out and reduced to rubble
WallCrumble	Chunk of rock being removed	A wall was mined enough to shake loose some ore
TorchOn	Torch being Lit	A torch was just lit
Footstep1	Footsteps	Someone walking, likely to be repeated multiple times in a row
Footstep2	^	^
Footstep3	^	^
Flap	Wing flapping movement	Something, likely a bat, moving by flapping its wings
WagonMove	Wagon Movement	A wagon carrying items slowly moving around
Alerted	Solid Snake alert	An enemy just spotted you, or you both just spotted eachother
StealthAlerted	Quiet alert	You or an ally spotted an enemy, but they havent seen you yet
HeatingUp	Smithy heating up	A smeltery is heating up to break ores into bars
Smelting	Smithy smelting ore	A smeltery is fully heated and is smelting ore
HammerClang	Anvil hammering	A blacksmith is working on tools on an anvil
PotionMix	Potions being brewed	An alchemist is mixing potions in a hut nearby
Training	Units being trained	A unit is training nearby, practicing sword techniques or whatever
YesMilord1	Generic acknowledgement	A unit was just selected, and you're setting up an order to be given
YesMilord2	^	^
YesMilord3	^	^
RightAway1	Generic acceptance of orders	You just finalized the order made above, and the unit responds to confirm
RightAway2	^	^
RightAway3	^	^
MapReveal	Unknown territoriy being revealed	You were shown a map, and get an animation of the new tiles being revealed
FireSpell	Firebolt spell	A fireball being launched
FireCrackle	Fireplace crackle	A torch or fireplace crackling
FireCrackle	Fireplace crackle	A torch or fireplace crackling
Burning	Immoation	A unit is burning, likely totally immolated
Whirlwind	Whirlwhind attack	You swing your weapon around yourself, hitting everyone nearby
HumanTalk1	Human talking	A human talking about anything, not real words, just sims-ish mumbles
HumanTalk2	^	^
HumanTalk3	^	^
GoblinTalk1	Goblin Talking	A goblin ^
GoblinTalk2	^	^
SpiderTalk	Spider talking	A Spider ^
CultTalk	Cultist talking	A cultist ^
Throw	Something being thrown	You throw something in a direction, could be anything
GlassShatter	Glass breaking	A glass bottle is broken, likely after being thrown
SpellCharge	Spell being charged up	A generic amgic spell is being charged
Fading	Becoming Invisible	A unit is fading into invisibility
Appearing	Becoming Visible	A unit is revealed from invisibility
Buff	Buffed Unit	A unit was buffed in some way
Debuff	Debuffed Unit	A unit was debuffed in some way
Eat	Eat some food	A unit just ate something
Drink	Drink a potion	A unit just drank something, likely a potion
EquipW	Equip a weapon	A unit just equpped a weapon
EquipA	Put on armor	A unit just equiped some form of armor
Pickup	Pick up an item	A unit just picked up an item from the ground
Drop	Drop an item	A unit just dropped an item on the ground
Give	Give an item	A unit just gave someone else one or more items

E2. Art assets

File name	Real name	Description
Ground tile textures		
StoneGround1	Rock Ground tile	Standard cave ground found almost everywhere
StoneGround2	^	^
StoneGround3	^	^
StoneGround4	^	^
StoneGround5	^	^
TilledGround	Farmable tile	More dirt-like ground suitable for growing cave plants
WoodGround1	Wood ground tile	Wooden ground tiles for old abandoned ruins
WoodGround2	^	^
WoodGround3	^	^
CorruptGround1	Corrupted ground tile	Cursed ground tiles for areas effected by black magiks
CorruptGround2	^	^
BioGround1	Biological ground tile	Fleshy zerg-creep-like ground tiles that appear bioluminesent
BioGround2	^	^
Walls / Water		
HardWall	Hard Wall	Unmineable walls that limit where units can go
HardWallFull	^	^
HardWallNoLeft	^	^
HardWallNoTopLeft	^	^
HardWallOnlyBotRight	^	^
SoftWall	Soft Wall	Mineable walls that drop nothing
SoftWallFull	^	^
SoftWallNoLeft	^	^
SoftWallNoTopLeft	^	^
SoftWallOnlyBotRight	^	^
OreWall	Ore Wall	Mineable walls that drop a single ore / gem
OreWallFull	^	^
OreWallNoLeft	^	^
OreWallNoTopLeft	^	^
OreWallOnlyBotRight	^	^
RichOreWall	Rich Ore Wall	Mineable walls that drop lots of ores / gems
RichWallFull	^	^
RichWallNoLeft	^	^
RichWallNoTopLeft	^	^
RichWallOnlyBotRight	^	^
GemWall	Gem Wall	Mineable walls that drop lots of gems
GemWallFull	^	^
GemWallNoLeft	^	^
GemWallNoTopLeft	^	^
GemWallOnlyBotRight	^	^
EternalWall	Eternal Wall	Mineable walls with an endless amount of ore
EternalWallFull	^	^
EternalWallNoLeft	^	^
EternalWallNoTopLeft	^	^
EternalWallOnlyBotRight	^	^
EmptyEternalWall	Empty Eternal Wall	Mineable wall that's slowly regenerating itself with ore
EmptyEternalWallFull	^	^
EmptyEternalWallNoLeft	^	^
EmptyEternalWallNoTopLeft	^	^
EmptyEternalWallOnlyBotRight	^	^

EmptyEternalWallOnlyBotRight	^	^
RuinsWall	Ruins Wall	Unmineable walls with an ancient ruin theme carved into them
RuinsWallFull	^	^
RuinsWallNoLeft	^	^
RuinsWallNoTopLeft	^	^
RuinsWallOnlyBotRight	^	^
Water	Water	Cavewater that only a few units can pass
WaterFull	^	^
WaterNoLeft	^	^
WaterNoTopLeft	^	^
WaterOnlyBotRight	^	^
Smoke	Smoke	Obscuring smoke that blocks vision of that square
Non-wieldable items		
Ore1	Ore	Rocks / minerals that can be smelted into bars and whatnot
Ore2	^	^
Ore3	^	^
IronBar	Iron bar	A smelted iron bar used in construction or smith work
SteelBar	Steel Bar	^ steel bar ^
MithrilBar	Mithril bar	^ mithril bar ^
BloodBar	Blood Bar	Magically constructed bar made of flesh
Goldbar	Gold Bar	Smelted gold bar just there to be valuable
Gem1	Gemstone	Gemstones used for magical or payment purposes
Gem2	^	^
Gem3	^	^
Gem4	^	^
Gem5	^	^
EmptyPot	Empty Potion	Empty potion that can be filled with whatever
RedPot	Colored Potion	Generic colored potion
GreenPot	^	^
BluePot	^	^
YellowPot	^	^
PurplePot	^	^
TealPot	^	^
BlackPot	^	^
BloodPot	^	^
Firecracker1	Firecracker	Firecracker or flashbang; pull and throw-like grenade
Firecracker2	^	^
Firecracker3	^	^
Firecracker4	^	^
Bomb	Bomb	Generic cartoonly primitive bomb, unlit
BombLit	Lit Bomb	Generic cartoonly primitive bomb, with a not lit fuse
HuntingNet	Hunting Net	Hunting net used to capture small prey
Plants1	Plants	Cavernous plants grown and used in potions and whatnot
Plants2	^	^
Plants3	^	^
Plants4	^	^
CorruptPlant	Corrupt Plant	Plants that have been corrupted by evil magicks
Log	Log	Generic log gained from cutting cave plants
Meat	Meat	Meat dropped from cave creatures, generic turkey stick is fine
RottenMeat	Rotten Meat	Rotten meat from when above meat is left unstored for too long

RottenMeat	Rotten Meat	Rotten meat from when above meat is left unstored for too long
Leather	Leather	Leather from some dead animal
Scroll1	Scroll	Mystic scroll that casts a spell when used
Scroll2	^	^
Scroll3	^	^
Scroll4	^	^
Scroll5	^	^
MessageScroll	Message Scroll	Paper with an intelligence report on it, or just a wraped up note
HumanCorpse	Human Corpse	Generic human corpse
GoblinCorpse	Goblin Corpse	Generic goblin corpse
SpiderCorpse	Spider Corpse	Generic spider corpse
MonsterCorpse	Monster Corpse	Generic monster corpse of anything else
SkeletonCorpse	Skeleton	Human / goblin corpse degraded to a generic skeleton
Bone1	Bone	Random bone left from a decayed body
Bone2	^	^
Bone3	^	^

<u>Wieldable-items</u>		
WoodPick	Pickaxe	Pickaxe used for mining
IronPick	^	^
SteelPick	^	^
MagicPick	^	^
WoodSword	Sword	Sword used for slicing
IronSword	^	^
SteelSword	^	^
MagicSword	^	^
WoodSpear	Spear	Spear used for thrusting
IronSpear	^	^
SteelSpear	^	^
MagicSpear	^	^
IronShield	Shield	Shield used for blocking
MagicShield	^	^
LeatherArmor	Leather armor	Armor used for not-dieing
PlateArmor	Plate armor	^
MagicPlateArmor	Enchanted plate armor	^
PlainRobes	Plain robes	Robes with no magical properties, yet
MagicRobes	Magical Robes	Wizardish robes, more tribal than fancy
RogueRobes	Rogueish stealth robes	Cloak-and-dagger-like robes
CultRobes	Cultist robes	Ominous cultist robes with hood
IronKnife	Knife	Knife used for stabbing
SteelKnife	^	^
MagicKnife	^	^
CultKnife	^	Knife used for stabbing a sacrifical sheep: tribal / bloody
Stick	Stick	Stick used for poking
Club	Club	Club used for smashing
Bludgeon	Bludgeon	Bludgeon used for knocking out guards with a hit to the back of the head
HookChain	Hook and Chain	Meat-hook and chain used for making sure enemies cant escape a-la Pudge
WoodBow	Bow and arrow	Generic bow with a notched arrow
IronBow	^	^
MagicBow	^	^
WoodStaff	Staff	Staff used for walking
MagicStaff	Magic staff	Staff used for casting spells, but obviously made from the above staff
SkullStaff	Skulled staff	Staff with a skull on top
TribalStaff	Tribal staff	Staff with a tiki mask and other baubles
BloodyStaff	Bloody Staff	Cultist staff with organs and such on it

BloodyStaff	Bloody Staff	Cultist staff with organs and such on it
OnTorch	Lit Torch	Torched used for lighting up the place
OffTorch	Unlit torch	Snuffed torch
OnLantern	Lit lantern	Lantern used for lighting up the place more
OffLantern	unlit lantern	Snuffed lantern
Orb1	Clear orb	Like a seer's crystal ball, only handheld
Orb2	Darkish orb	Above, but kinda dirty-looking inside
Orb3	Darkening orb	Above, but mostly dirty looking
Orb4	Corrupt orb	Completely darkened orb, cultist-y

Hero		The guy the player will be seeing every moment
HeroIdle	Idle hero	The hero standing still
HeroInjured	Injured Hero	The hero idle but injured, used instead of idle whenever at low hp
HeroAlert	Alert Hero	The hero in a ready stance when an enemy is on screen / in combat
HeroCorrupt	Corrupted hero	The hero idle but with cracks and whatnot like a cultist, maybe glowy red eyes
HeroAttackMelee	Swinging hero	The hero swinging his melee weapon
HeroAttackRange	Bow-shooting hero	The hero shooting a bow at someone
HeroThrowing	throwing hero	The hero throwing something
HeroCastHand	Spellcasting hero	The hero casting a spell
HeroCastItem	Item-using hero	The hero using a magical item, just holding a scroll above his head is fine
HeroRecoil	Recoiling hero	The hero recoiling after just getting hit by an attack

Wielding units		
GobArcher	Goblin archer	The goblin's designated ranged-weapons user
GobHunter	Goblin hunter	Goblin in charge of catching cookable prey for meat
GobCaptain	Goblin captain	Authoritative goblin but not that much stronger than normal
GobScout	goblin scout	Goblin made for exploration and sneaking around
GobBoss	goblin boss	Strong goblin leader
GobDigger	goblin digger	Goblin peon made for mining and construction
GobFighter	goblin fighter	Goblin grunt in charge of punching / clubbing things
GobWitch	goblin witch	Witch-doctor like goblin with a tribal mask
Cultist	Generic cultist	Crazed generic-bloody cultist most likely wielding a knife
WitchCultist	Runed cultist	Spellslinging demon-summoning Cultist with glowing bloody runes all over his body

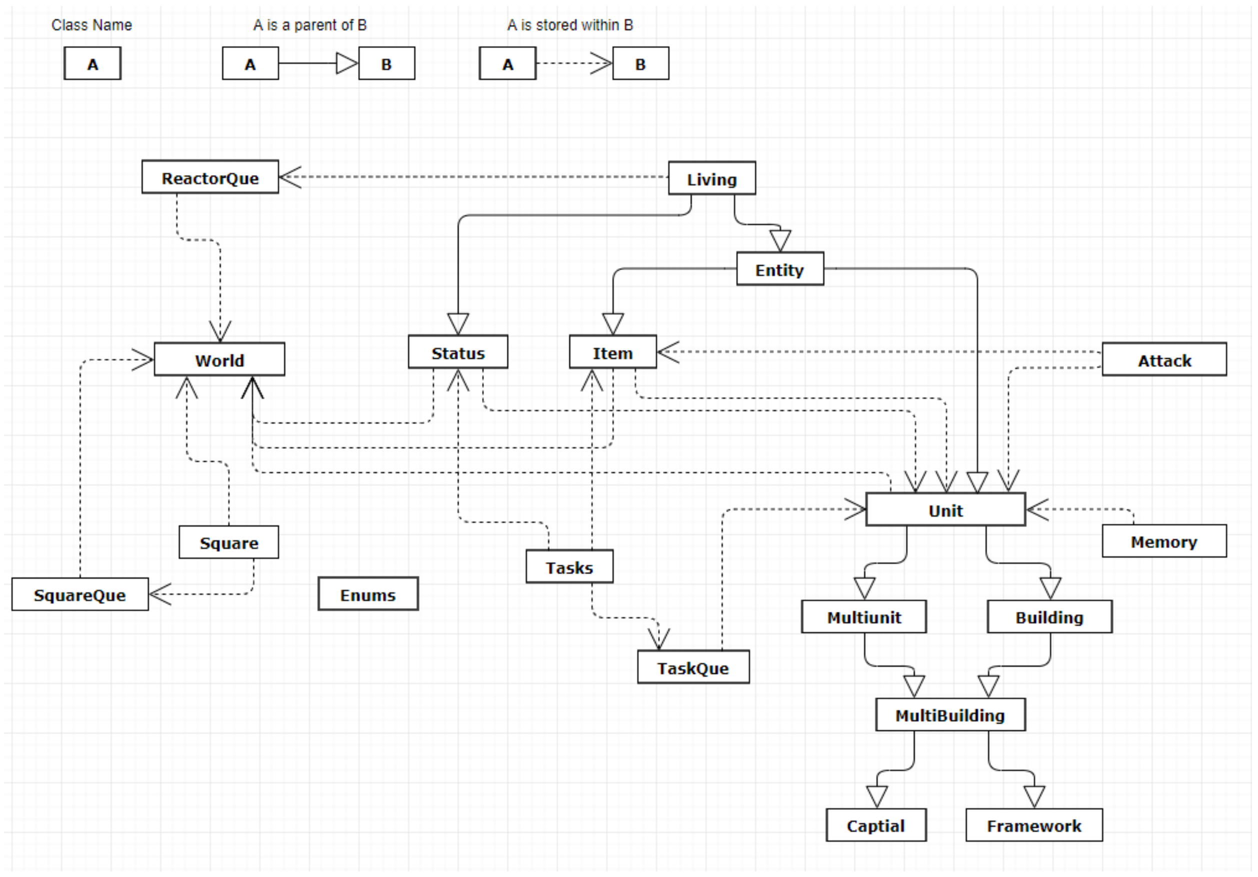
Goblin buildings		
Smeltery	Goblin smeltery	Smoke and rock oven that could turn ores into slag
Barracks	goblin barracks	goblin housing / war room area
CraftArea	goblin crafting station	Building with lots of low-quality crafting supplies / anvils for weaponmaking
GobStorage	goblin storage area	Storage building or sectioned area for lots of supplies
WitchHut	goblin witch hut	Spooky area filled with skulls leading to a mystic oracle-like building
GobCapital	goblin capital	Slightly-fancy tent with gold and supplies and message boards similar to human capital
GobBarricadeUD	goblin barricades	Spears and sticks and such formed into semi-fence structures
GobBarricadeLR	goblin barricades	^

Other units		
Spider1	Spider	Giant feral spider, just a human-sized tarantula
Spider2	Hauling spider	Same tarantula, only dragging someone it's captured
Seeker	Seeker spider	Thinner spider used for scouting, daddy-longlegs like
Spiderlings	Spiderlings	Mass of baby spiders
BigSpider	Boss spider	Older, battle scarred spider, obviously stronger than other spiders
Spidernest	Spidernest	Giant egg-sac nest with spiderlings crawling around
Spiderhole	Spider hole	A burrowed-spider waiting to pop out
Mole	Giant Mole	Massive feral mole, digging claws and possibly rabid-looking
Ogre	Ogre	Giant club-wielding brute
Golem	Magic golem	Stone / runed golem, big and intimidating tank, not cultist-y
BigGolem	Massive golem	Same golem, only 4x4
Zombie	Human zombie	Rotten, skeletal human zombie
Wisp1	Wisps	Orb-like luminous wisps
Wisp2	Wisps	^
Wisp3	Wisps	^
HumCartFull	Filled human cart	Cart filled with supplies, pullable by humans (not horse-drawn carriage)
HumCartEmpty	empty human cart	same cart above only visibly empty
GobCartFull	filled goblin cart	Goblin cart filled with supplies, more rickshaw than human cart
GobCartEmpty	empty goblin cart	Empty goblin cart
Bat	Bat	Meaty bat that flaps around blindly and gets killed for meat
Batcave	Bat-cave	Hole-filled wall where bats come out every once in a while
Rat	Rat	Tiny rat, barely noticable
Critter	Critter	Miniature non-threatening dog-like critter that'll wander aimlessly for decoration
Slither	Slither	Like magic-the-gathering slither, 2 claws for hands and sharp beak, no legs, small and non-threatening
PressureOn	Activated Pressure plate	Pressure-plate that has something heavy on it
PressureOff	Deactivated pressure pl	Pressure-plate that's yet to be stepped on
LeverOn	Lever pulled down	Lever that's been pulled to one side
LeverOff	upright lever	Upright lever that hasn't been used
WoodDoorOpen	Open door	Opened door, with the hinges on the left / right
WoodDoorClosed	closed door	closed door
MetalDoor***	Metal door	Metalic-door made of harder materials
RuinsDoor***	Ruins-themed door	Ancient fancy-door connected to ruin-walls on the left / right
SpikeOn	Spike-trap extended	Spike-trap with spears sticking out from the ground piercing whoever's above
SpikeOff	Spike-trap un-activated	The holes of above spear-trap without the spears sticking out, yet
FlamesOn	Activated Flame-trap	Flames jutting out of holes on the ground
FlamesOff	Unused flametrap	The holes of said flames above, without the flames
OrbOn	Orb-turret activated	An orb on a pedestal that's probably going to zap anyone who comes near
OrbOn	Orb-turret activated	An orb on a pedestal that's probably going to zap anyone who comes near
OrbOff	Orb-turret deactivated	An orb on a pedestal that appears deactived compared to above
OrbEmpty	Orb-turret without an o	The pedestal from above with no orb, its been stolen
Portal	Teleportation portal	A portal for teleportation, should look jump-in-able
BookshelfFull	Filled bookshelf	An old bookshelf filled with books
BookshelfOne	Single-book bookshelf	An old bookshelf with a single important looking book
BookshelfEmpty	Empty bookshelf	An empty bookslef, slightly ruined and decrepid
Rune	Glowing rune	A cultisty rune glowing some random color
LargeRune	Big glowing rune	Same as above, only 2x2 in size
BigRune	Masive glowing rune	Same as above, only 3x3 in size and the middle made for sacrifices
LaserUD	Mystical Laser lines	A line filled with magical energy, like a spell shooting from across the square
LaserLR	^	^
LaserUL-DR	^	^
LaserUR-DL	^	^

Command Icons		All are spell / command icons that may be used for more than one relevant ability
Interact	Interact with x	A very generic 'interaction'. Could be anything from pulling a lever to giving items to a building
Talk	Talk with someone	Talk with another unit
Trade	Trade items	Trade items with a unit or building
Scout	Go scouting	Scout to some far-off position
AttackPoint	Attack-move towards a	Attack anything you see while moving to a position
AttackGuy	attack a specific unit	Attack a specific unit in view
GiveMessage	Deliver a message	Deliver a message to an allied building
Construct	Construct a building	Order the construction of a building
Grab	Grab a unit	Grab and carry someone in view
Release	Release a grabbed unit	Release anyone you're currently grabbing
Follow	Follow a unit	Follow someone in view
Lead	Lead in front of a unit	Guide someone wherever they go, trying to stay in front of them
Use	Use an item	Use an item you're holding
Equip	Equip a weapon	Equip an item they're holding
Give	Give an item	Give an item to another unit or building
Drop	Drop an item	Drop an item they're currently holding
Throw	Throw an item	Throw an item at someone or someplace in view
Inspect	Inspect something	Inspect something, bringing up its info-screen usually
Knockout	Knockout a unit	Try to knockout a unit non-lethally
Wait	Wait here	Wait at a targeted position until I say otherwise
WaitXTurns	Wait here for x turns	Wait at a targeted position for x turns
Return	Return to me	Return to me when you're done with the previous command
GetReturn	Get info then return	Get updates from an information hub, then return to me
DoFlee	If in trouble, flee	If you run into enemies along the way, just run to safety
DoInjuredRetreat	If in trouble, put up a fig	If you run into enemies along the way, put up a fight but run if you're low on hp
DoFight	If in trouble, fight to the	If you run into enemies along the way, fight them till one of you dies
AndThen	And then,,,	Queue up commands
MassFlee	Everyone flee	Everyone around me should run away
MassAttack	Everyone attack	Everyone around me should attack to some point
MassFocusfire	Everyone attack this guy	Everyone around me should focus attacks on that guy
MassFollow	Everyone follow me	Everyone around me should start following targeted unit
MassLead	Everyone lead in front o	Everyone around me should start leading in front of targeted unit
MassHold	Everyone hold position	Everyone around me should hold position where they're currently at
MassHoldMe	Everyone hold position	Everyone around me should get as close as possible to me and then wait
Spell Icons		
Whirlwind	Whirlwind	A spin-your-arms-around attack that hits everyone around you
Shove	Shove	A spin-your-arms-around attack that shoves away everyone around you
Heavypunch	Heavy attack	A heavy attack that deals more damage but slows you down
Knockpunch	Knockback punch	A weapon or punch that knocks back the target
Interrogate	Interrogate	Interrogate a captive enemy for information they know
Fireball	Fireball	Cast a fireball on someone
Freeze	Freezing blast	A chilling aoe blast that freezes enemies hit
Flameblast	Firey blast	A Fiery blast that ignites enemies hit
Smokegrenade	Smoke grenade	A smoke grenade that can obstruct vision
Guard	Guard	Guard an incoming attack
Block	Block	Block with a shield
Shieldsup	Shields up!	Form a shieldwall with a massive tower-shield
Counterattack	Counterattack	Prepare to reflect an attack and counter with your own
Throwdust	Throw dust	Throw dust at enemies eyes, blinding them
Bless	Bless	A holy blessing on a friendly unit
Curse	Curse	A debilitating curse on an enemy unit
Moralboost	Moral boost	A Fistpump that improve allied stats
Commandpoint	Commanding point	A commanding-looking point, as generic as possible
Heal	Heal	A healing spell
Regeneration	Regeneration	A heal-over-time spell
Cauterize	Cauterize	Heal wounds by burning them shut
Knifethrow	Knife throw	Throw a knife at a target
PKnifethrow	Poisoned knife throw	Throw a poisoned knife at a target
BKnifethrow	Bloody knife throw	Throw a bloody knife at a target to make them bleed over time

Status effect icons		
Scared	Scared	This unit is scared of something
Terrified	Terrified	this unit is REALLY scared of something
Cowardly	Cowardly	This unit wants to run away from battle
Selfish	Selfish	This unit only cares about himself
Alerted	Alerted	This unit is aware of enemies nearby
Cautious	Cautious	This unit is suspicious that there may be nearby enemies
Safe	Safe	This unit is unaware of nearby enemies
Fading	Fading	This unit is becoming invisible
Invisible	Invisible	This unit blends into the background and cant be seen
Revealed	Revealed	This unit is revealed, even if invisible
Numbing	Numbing	This unit is going numb, slowly becoming unable to move
Paralyzed	Paralyzed	This unit is paralyzed and can't move at all
Helpless	Helpless	This unit is helpless for some reason or another, and isn't a threat
Threatening	Threatening	This unit looks like a dangerous threat
Hooked	Hooked	This unit has been caught in a hook and has its movement limited
Ignited	Ignited	This unit is on fire
MagicIgnited	Ignited with magic	This unit is on fire with magical, multicolor flames
HolyIgnited	Ignited with holy flame	This unit is on fire with blessed holy flames
DarkIgnited	Ignited with cult flames	This unit is on fire with dark, unholy, cultish flames
Illuminated	Illuminated	This thing is glowing
Darkened	Darkened	This thing is much darker than it usually is, sucking light around it
Disguised	Disguised	This unit is disguised, wearing a mask a-la tf2 to trick enemies
Marked	Marked	This thing is marked, making it something to focus on
Tracked	Tracked	This unit is being tracked, making it easier to follow
Asleep	Asleep	This unit is asleep
Nightmares	Nightmares	This unit is asleep and having nightmares
Netted	Netted	This unit is caught in a net
Growing	Growing	This unit is growing to a larger size
Shrinking	Shrinking	This unit is shrinking to a smaller size
AttackBoost	Attack Boost	This thing has a physical attack buff
AttackLoss	Attack Loss	This thing has a physical attack debuff
PoisonAttack***	Poison attack boost/los	This thing has a posion attack buff or debuff
MagicAttack***	Magic attack boost/loss	This thing has a magic attack buff or debuff
PureAttack***	Pure attack boost/loss	This thing has a pure attack buff or debuff. Pure damage is only blocked by pure defence, so its mostly piercing
Speed***	Speed boost/loss	This thing has a speed buff or debuff
Speed***	Speed boost/loss	This thing has a speed buff or debuff
Defence***	Defence boost/loss	This thing has a physical defence attack buff or debuff
PoisonDefence***	Poison defence boost/l	This thing has a posion defence buff or debuff
MagicDefence***	Magic defence boost/lo	This thing has a magical defence buff or debuff
PureDefence***	Pure defence boost/los	This thing has a pure defence buff or debuff
Courage	Courage	This unit is encouraged and brave
Blinded	Blinded	This unit is completely blind
Farsight	Farsight	This unit can see farther than normal
Darksight	Darksight	This unit can see farther in the dar than normal
Nightvision	Nightvision	This unit can see everything in the dark
Truevision	Truevision	This unit can see invisible units
Xrayvision	Xrayvision	This unit can see through walls
Shortsighted	Shortsighted	This unit has shorter vision than usual
MagicProtect	Magic Protect	This unit is protected against harmful spells
MagicImmune	Magic Immunity	This unit is immune to all spells
MagicVulnerable	Magic Vulnerable	This unit is vulnerable to spells
Poisoned	Poisoned	This unit is poisoned
Bleeding	Bleeding	This unit is bleeding
Insane	Insane	This unit is crazed and insane
Enranged	Enranged	This unit is in a battle frenzy, not strictly bad
Chainlightning	Chainlightning	This unit is effected by lightning that will spread to other units around it
Webbed	Webbed	This unit is caught in a spider's web

Appendix F: Class hierarchy summary



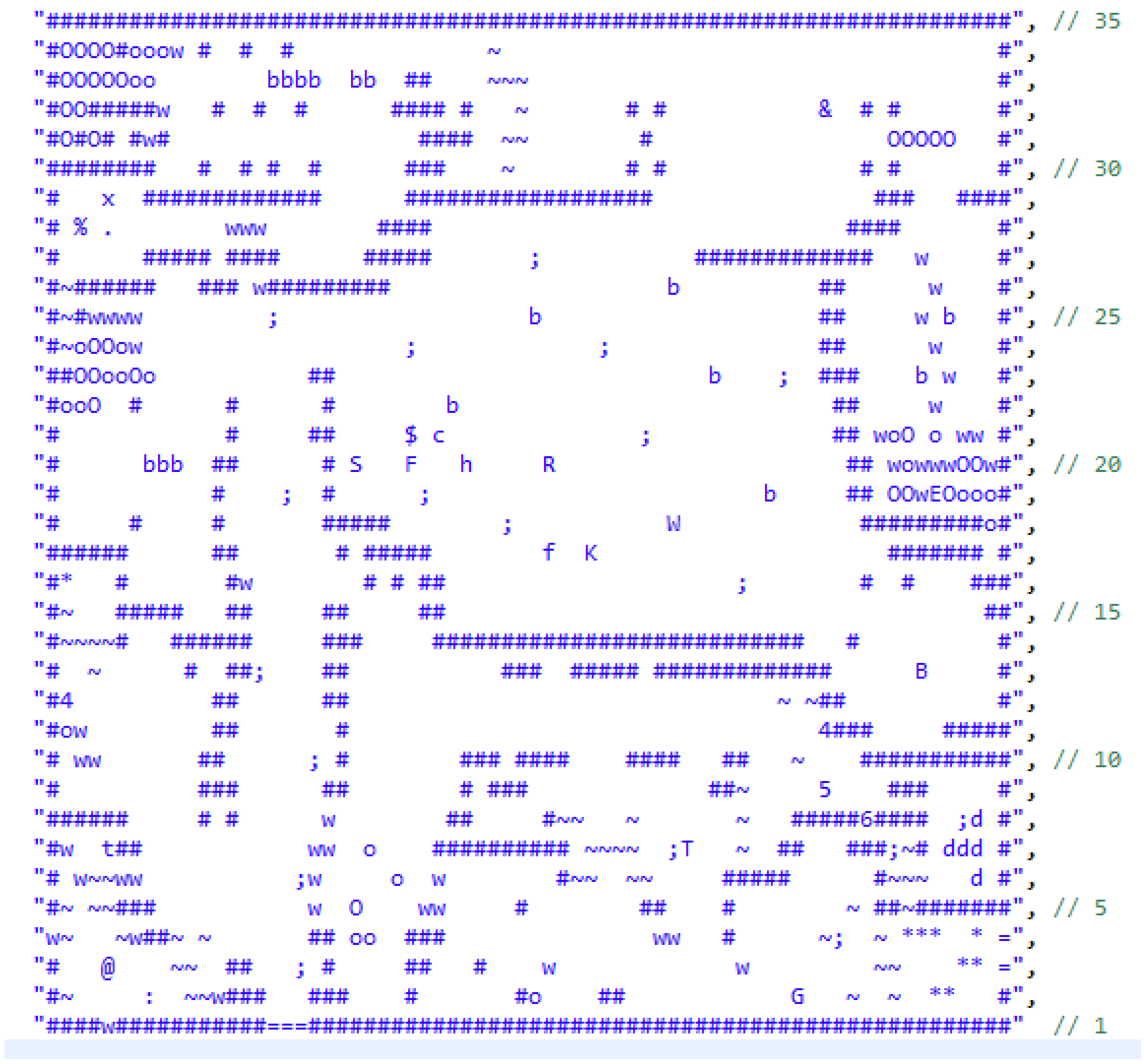
Appendix G: Level maps

G1. Map key

Key: the character below will produce the resulting unit or item in the designated x/y coordinate

w = Wall	b = Bat	d = Digger	& = Storage
o = Orewall	B = Batcave	c = Cart	$ = Stronghold
O = Rich Ore Wall	T = Troll	s = Scout	S = Smeltery
E = Eternal Wall	Z = Monstrosity	h = Hunter	R = Barracks
# = Hard Wall	/ = Eyebeast	f = Fighter	K = Blacksmith
= = Floorwall		p = Priest	W = Witchhut
~ = Wood blockage	x = Spider	C = Captain	; = Torchstand
A = immortal wall	% = Spider Nest	e = Explorer	
	* = Spiderling	a = Archer	[= Trap
@ = Player	. = Seeker		l = Lever
! = Torch	) = Stalker	H = Human	D = Door
: = Lantern			
t = Copper Sword	4 = Chest with goodies		
	5 = Chest with more goodies		
F = Tutorial Fighter 1	6 = Locked chest with goodies		
G = Tutorial Fighter 2	7 = Chest with rare goodies		

G2. Tutorial level 1

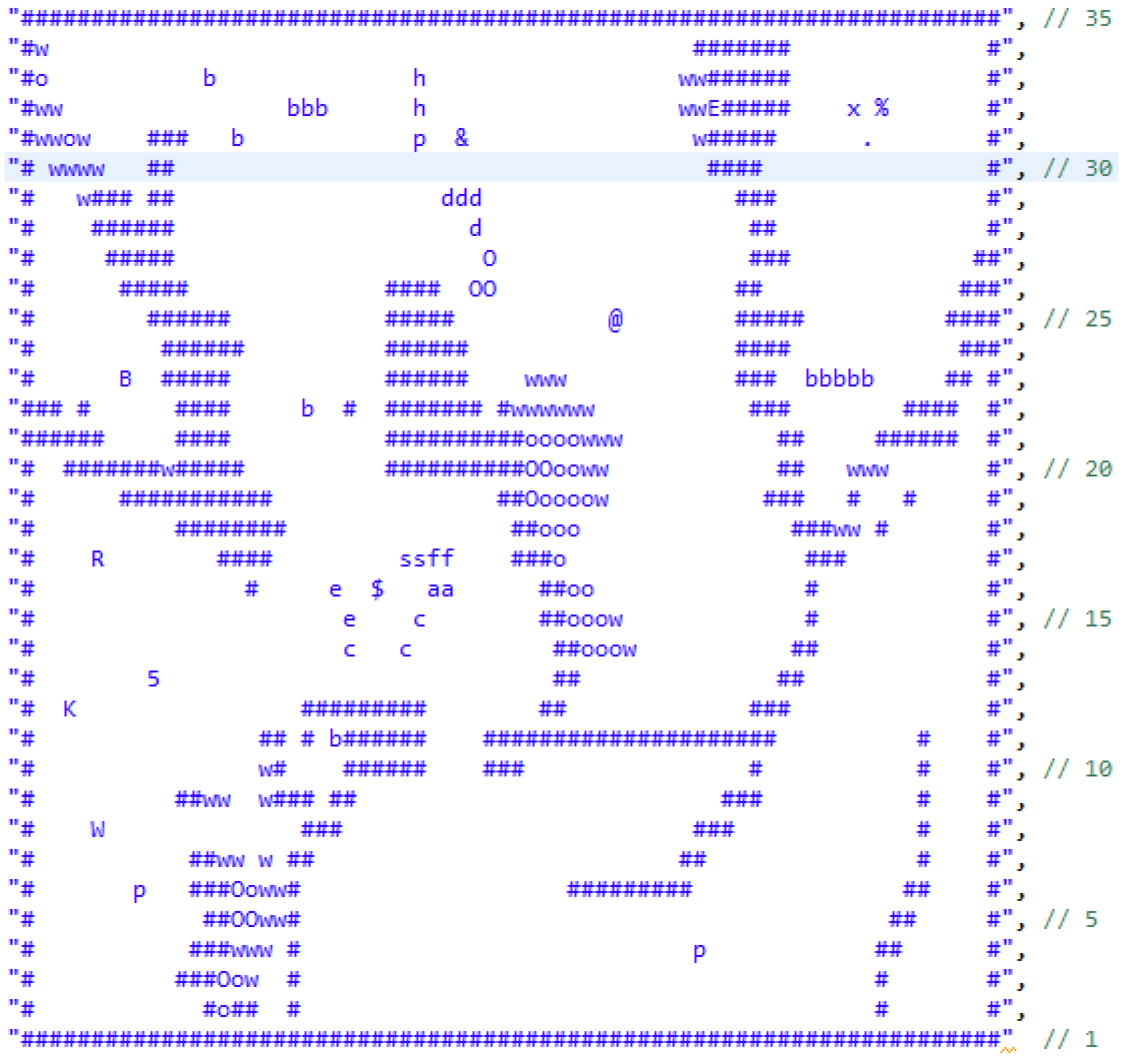


```
"######################################################################", // 35
"#OOOO#ooow #  #  #              ~                                    #",
"#OOOOOoo        bbbb  bb  ##    ~~~                                  #",
"#OO#####w   #  #  #      #### #   ~       # #           &  # #       #",
"#O#O# #w#                  ####  ~~        #                 OOOOO   #",
"########   #  # #  #      ###    ~        # #              # #       #", // 30
"#   x  #############      ##################                ###   ####",
"# % .        www        ####                              ####       #",
"#      ##### ####      #####       ;           #############   w     #",
"#~######   ### w#########                    b          ##      w    #",
"#~#wwww         ;                  b                    ##     w b   #", // 25
"#~oOOow                   ;             ;               ##      w    #",
"##OOooOo           ##                           b    ;  ###    b w   #",
"#ooO  #      #      #        b                           ##     w    #",
"#            #     ##     $ c              ;             ## woO o ww #",
"#      bbb  ##      # S   F   h     R                     ## wowwwOOw#", // 20
"#           #    ;  #      ;                        b     ## OOwEOooo#",
"#     #     #       #####        ;           W             #########o#",
"######      ##       # #####        f  K                     ####### #",
"#*   #       #w        # # ##                     ;        #  #    ###",
"#~   #####   ##     ##     ##                                       ##", // 15
"#~~~~#   ######     ###     ###########################   #          #",
"#  ~      #  ##;    ##           ###  ##### #############      B     #",
"#4          ##      ##                               ~ ~##           #",
"#ow         ##       #                                  4###     #####",
"# ww       ##      ; #        ### ####    ####   ##   ~    ###########", // 10
"#          ###      ##        # ###             ##~     5    ###     #",
"######     # #      w        ##     #~~   ~       ~   #####6####  ;d #",
"#w  t##            ww  o    ########## ~~~~  ;T   ~  ##   ###;~# ddd #",
"# w~~ww           ;w     o  w        #~~  ~~     #####      #~~~   d #",
"#~ ~~###           w  O    ww     #        ##    #        ~ ##~#######", // 5
"w~   ~w##~ ~       ## oo  ###               ww   #      ~;  ~ ***  * =",
"#   @    ~~  ##   ; #     ##   #    w             w         ~~    ** =",
"#~     :  ~~w###   ###    #       #o    ##            G   ~  ~ **    #",
"####w###########===###################################################"  // 1
```

The player (@) starts in the lower left corner, waking up after having been knocked out due to an earthquake, and are reminded of their duty to check up on a base to the north. Some starting equipment can be found by groping through the darkness, and after breaking through the rubble they meet a fighter who welcomes them. His memories of the base are shared, but the road turns out to be blocked by fallen rubble. A miner is needed to pass, where they're told of a small mining operation to the east. Heading that way they're warned of a troll (the T) up ahead by a fleeing fighter, who turned off his lantern to escape unseen. If the player turns off their lantern they can sneak through the rubble to the south to reach the base, or head to the right to access a few chests (4, 5, 6) for better equipment. Leaving their light on puts them in range of the troll's vision, who will come and fight them.

Coming up to the base reveals a swarm of spiderlings, who must be cleared out to reach the miners trapped behind a wall of rubble (bottom right). The player can then lead the miners back along the path, breaking down the walls in their way and reaching the stronghold (Middle section, $). They're informed of the situation of suspiciously inconvenient quakes and of spiders starting to get aggressive, with their next goal being clear out any nests they can find. They have ample time to mine as much as they need to make new equipment, raise a small army and scout out the nearby spider nest, which will start sending out hunters if enough time passes. After the nest is cleared they're informed they've beaten the tutorial, and can quit out or explore the rest of the map if they please.

G3. Planned tutorial level 2



```
"######################################################################", // 35
"#w                                              #######              #",
"#o           b              h                  ww######              #",
"#ww                bbb      h                  wwE#####    x %       #",
"#wwow    ###   b            p  &               w#####      .        #",
"# wwww   ##                                      ####                #", // 30
"#   w### ##                   ddd                  ###               #",
"#    ######                     d                   ##               #",
"#     #####                      O                  ###             ##",
"#      #####              ####  OO                 ##              ###",
"#        ######           #####           @        #####          ####", // 25
"#         ######          ######                   ####            ###",
"#      B  #####           ######    www            ###  bbbbb     ## #",
"### #      ####     b  #  ####### #wwwwww           ###        ####  #",
"######     ####           ##########oooowww           ##     ######  #",
"#  #######w#####          ##########OOooww            ##   www       #", // 20
"#      ###########                ##Ooooow           ###   #   #     #",
"#          #########               ##ooo               ###ww #       #",
"#    R        ####         ssff    ###o                 ###          #",
"#               #     e  $   aa      ##oo               #            #",
"#                      e    c        ##ooow             #            #", // 15
"#                      c   c          ##ooow          ##             #",
"#        5                            ##              ##             #",
"#  K                #########        ##              ###              #",
"#                ## # b######    #####################          #    #",
"#                w#    ######    ###                #           #    #", // 10
"#          ##ww  w### ##                          ###           #    #",
"#    W              ###                         ###             #    #",
"#           ##ww w ##                          ##               #    #",
"#       p   ###Ooww#                   #########               ##    #",
"#            ##OOww#                                          ##     #", // 5
"#           ###www #                            p            ##      #",
"#          ###Oow  #                                         #       #",
"#            #o##  #                                         #       #",
"######################################################################"  // 1
```

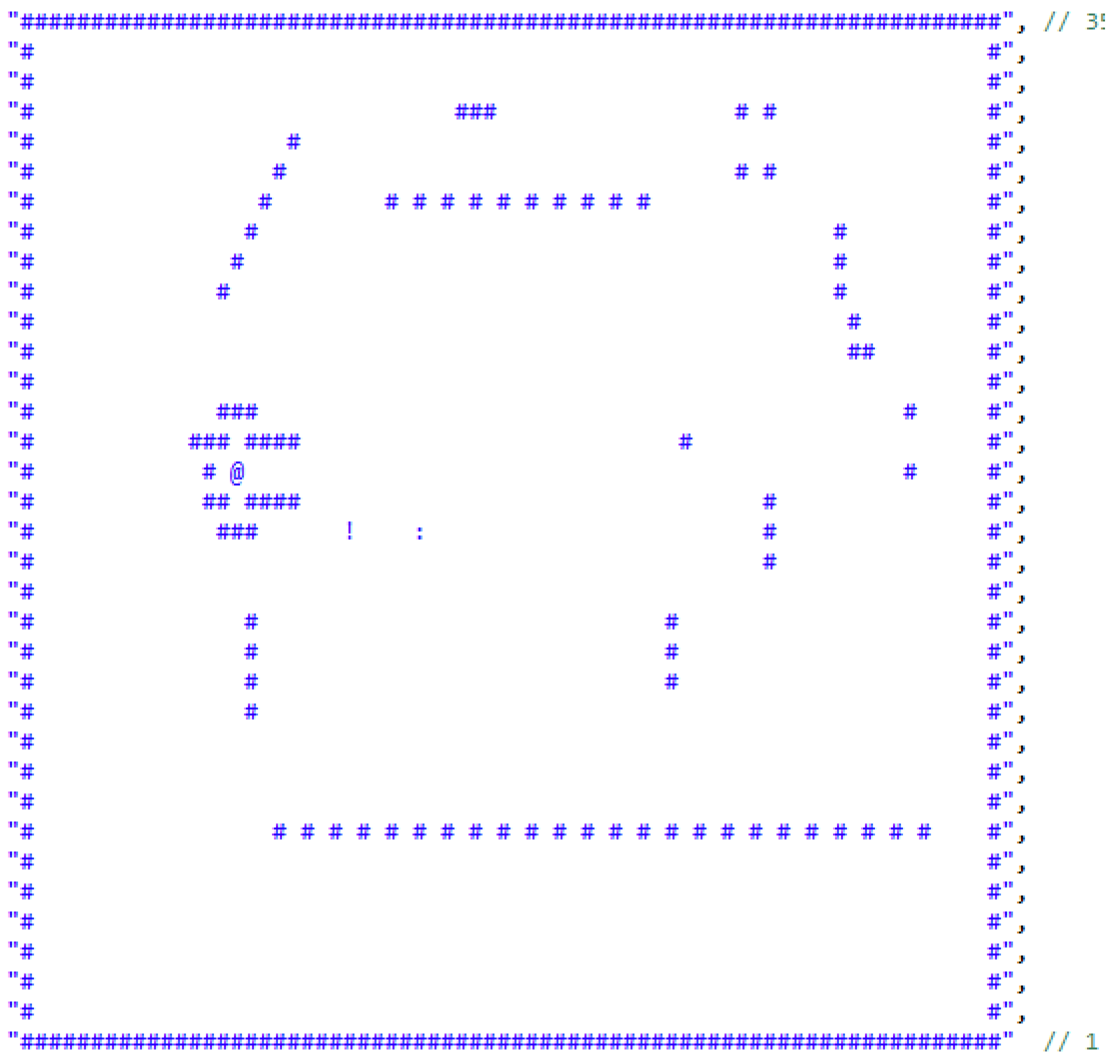
Prototype of the second planned level, made to introduce base-building mechanics and spider ambush tactics. The player was to be in charge of constructing buildings destroyed from earthquakes, and eventually learn of spiders picking off diggers and tasked with exterminating them. Condensed into one level to facilitate testing.

G4. Planned tutorial level 3

```
"######################################################################", // 35
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                         #                                          #",
"#                                                                    #",
"#                                              ######                #",
"#     #                                        #    #                #",
"#wwwwww                        &                  E #                #",
"#ooOOow                                        #    #                #",
"#                            dddp              ######                #",
"#ooO  #                                                              #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#     #                                                              #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"############# ##                                                     #",
"#          #   #######################                               #",
"#   ###### # # #                                                     #",
"#    p   # # # #                                                     #",
"#        # # # #       h  c                                 ####~#####",
"#        #                             K                    #        #",
"#######  # # # #         $                                  #        #",
"#bbbbb#  # # # #                                            #        #",
"#bbbbb#  # # # #        f f                      ############        #",
"#     #  # # # #                                 #          #   *****#",
"### ###  # # # #                       5         #       B  #*****   #",
"#     #! # # # #   W       R                     #          #  ***   #",
"#     @  #   #                                              #  ***   #",
"######################################################################"  // 1
```

Rough prototype of the third tutorial level, made to teach more about tasks, commands, memories and not believing everything one sees. The player would start in a small base, tasked with collecting three special talismans in nearby ancient ruins. The ruins themselves contained various traps and monsters trying to drive them away, and as the talismans get collected new monsters would start spawning periodically. They'd rush out and cause production-stopping chaos among the workers at home, involving hallucinations, madness and forgetfulness depending on the order the amulets were acquired. Collecting them all would attract a demon from a final sealed ruin, hunting the player relentlessly till they brought the amulets to its now opened altar.

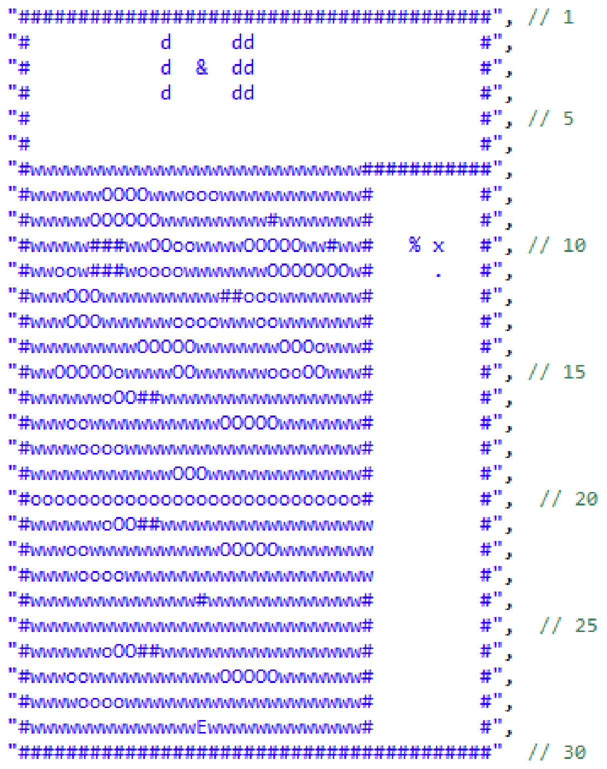
G5. Raycast test room



```
"######################################################################", // 3
"#                                                                    #",
"#                                                                    #",
"#                              ###                 # #               #",
"#                  #                                                 #",
"#                 #                                # #               #",
"#                #        # # # # # # # # # #                        #",
"#               #                                         #          #",
"#              #                                          #          #",
"#             #                                           #          #",
"#                                                          #         #",
"#                                                          ##        #",
"#                                                                    #",
"#             ###                                              #     #",
"#           ### ####                           #                     #",
"#            # @                                               #     #",
"#            ## ####                                 #               #",
"#             ###      !    :                        #               #",
"#                                                    #               #",
"#                                                                    #",
"#               #                             #                      #",
"#               #                             #                      #",
"#               #                             #                      #",
"#               #                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                 # # # # # # # # # # # # # # # # # # # # # # # #    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"#                                                                    #",
"######################################################################"  // 1
```

A debugging room made to test and experiment with how vision was drawn. The player was given infinite sight range, making long-distance blockers appear to cast shadows. The raycasting function went through numerous iterations before an acceptable method was confirmed to work.

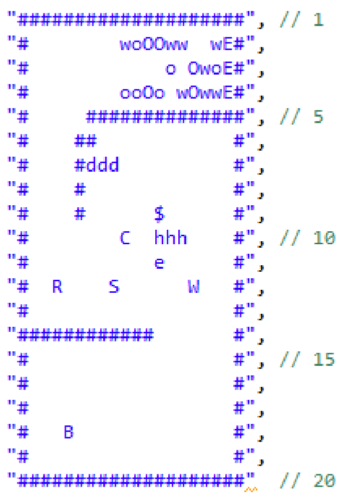
G6. Digger Room



```
"########################################", // 1
"#           d     dd                   #",
"#           d  &  dd                   #",
"#           d     dd                   #",
"#                                      #", // 5
"#                                      #",
"#wwwwwwwwwwwwwwwwwwwwwwwwwwww###########",
"#wwwwwwOOOOwwwooowwwwwwwwwwww#         #",
"#wwwwwOOOOOOwwwwwwwww#wwwwwww#         #",
"#wwwww###wwOOoowwwwOOOOOww#ww#   % x   #", // 10
"#wwoow###woooowwwwwwwOOOOOOOw#     .   #",
"#wwwOOOwwwwwwwwww##ooowwwwwww#         #",
"#wwwOOOwwwwwwoooowwwoowwwwwww#         #",
"#wwwwwwwwwOOOOOwwwwwwwOOOowww#         #",
"#wwOOOOOowwwwOOwwwwwoooOOwwww#         #", // 15
"#wwwwwwoOO##wwwwwwwwwwwwwwwww#         #",
"#wwwoowwwwwwwwwwwOOOOOwwwwwww#         #",
"#wwwwoooowwwwwwwwwwwwwwwwwwww#         #",
"#wwwwwwwwwwwwOOOwwwwwwwwwwwww#         #",
"#oooooooooooooooooooooooooooo#         #",  // 20
"#wwwwwwoOO##wwwwwwwwwwwwwwwwww         #",
"#wwwoowwwwwwwwwwwOOOOOwwwwwwww         #",
"#wwwwoooowwwwwwwwwwwwwwwwwwwww         #",
"#wwwwwwwwwwwwww#wwwwwwwwwwwww#         #",
"#wwwwwwwwwwwwwwwwwwwwwwwwwwww#         #",  // 25
"#wwwwwwoOO##wwwwwwwwwwwwwwwww#         #",
"#wwwoowwwwwwwwwwwOOOOOwwwwwww#         #",
"#wwwwoooowwwwwwwwwwwwwwwwwwww#         #",
"#wwwwwwwwwwwwwwEwwwwwwwwwwwww#         #",
"########################################"  // 30
```

Simple room filled with walls of all types to test mining mechanics. The right side of the room was eventually added to make sure spiders were working as intended after a minor overhaul of the way status effects were handled.

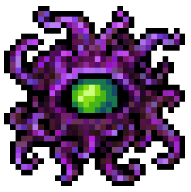
G7. Stress testing room



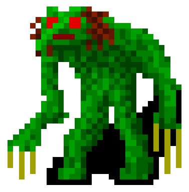
```
"####################", // 1
"#        woOOww  wE#",
"#            o OwoE#",
"#        ooOo wOwwE#",
"#     ##############", // 5
"#    ##            #",
"#    #ddd          #",
"#    #             #",
"#    #      $      #",
"#        C  hhh    #", // 10
"#           e      #",
"#  R    S      W   #",
"#                  #",
"############       #",
"#                  #", // 15
"#                  #",
"#                  #",
"#   B              #",
"#                  #",
"####################"  // 20
```

A miniature base with all unit-producing buildings, made to see how many could be handled at once on maximum turn-speed before causing lag. Miners and hunters continue to bring resources to the stronghold, which eventually assigns new workers to be built what speed up production, to the point the world is completely filled with kobold troops. Some of those troops blocked allied movement, so production would eventually grind to a halt.

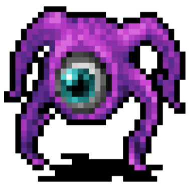
Appendix H: Unit list



Monstrosity. Scary looking, but actually very weak. Sneaks around in the dark and magically scares everyone nearby when confronted to make enemies prioritize running away. Runs away itself the moment it starts taking damage.



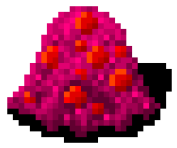
Troll. Regenerates health rapidly, but does low damage. Has terrible night vision, despite living in a cave.



Eyebeast. Weak in combat, but can gaze at targets to make them think everyone besides the eyebeast is an enemy.

Spider Seeker. Scouts of the spider army. Can become invisible for short periods of time, but very cowardly if visible.

Spider Hunter. A fully grown spider. Hunts weaker targets with invisibility and poison. Will drag paralyzed targets back to their nest, but flee from most combat situations.



Spider Nest. A huge nest of spiders, and technically their capital. Collects meat from hunted prey and spawns spiderlings over time.

Spiderling. A young and relatively tiny spider, but much more aggressive than its adult version. Will defend its nest and grow into other spiders if enough meat is hoarded.

Spider Stalker. A spider made for fighting. Injects a long lasting poison into targets then attempts to flee to darkness.

Kobold Captain. A kobold commander, in charge of leading the troops to battle enemies. If any are discovered, will rally a large army and hunt them down, but if the battle appears lost will order a call for retreat and run back to the capital to heal. A player will typically possess one of these as their avatar, overriding their usual behaviors.

Kobold Cart. A kobold hauler, being able to carry many more items than usual and used to deliver supplies en-masse between buildings.

Kobold Digger. A kobold miner, able to use picks. Can only carry a few ores at a time before having to go store them at the closest storage area or capital.

Kobold Explorer. A kobold with better night-vision than normal. Is sent off to explore the unknown, coming back when enough new discoveries are made.

Kobold Fighter. The staple of the kobold army, able to use a variety of weapons and armor. Has much better combat stats than others, and can be trained at a barracks to learn new techniques.

Kobold Archer. A kobold that prefers to use a bow, staying away from enemies. Is only a little stronger than the average kobold and can't see that far in the dark, making his range only useful in certain situations.

Kobold Hunter. A kobold hunter-gatherer whose preferred target is bats. Has a bit of combat training, but is still a target for spiders.

Kobold Priest. A kobold magic user, able to cast HolyFire on enemies and passively has truesight to reveal invisible things. Can wield magic staffs and is great against spiders.

Kobold Scout. A kobold with better light-vision than normal. Explores areas nearby around the capital, usually being the first watchdog to report an attack.

Kobold Barracks. Kobold training grounds and unit producer, spawning troops if given enough metal and meat.

Kobold Blacksmith. A kobold crafting area to turn iron or steel bars into weapons, armor or tools.

Kobold Smeltery. The place where ores are smelted into bars. The process of heating up the smelting furnace takes some time, but when its on ores are quickly processed.

Kobold Storage. A kobold storage facility for collecting unprocessed goods. Usually constructed near mining and hunting zones, where harvesters are saved from walking all the way back to the capital with each trip. Will also hold tool upgrades and give them out to the appropriate user when they next arrive.

Kobold Stronghold. A kobold fortress, and their capital. The central hub of a kobold base that directs anyone who interacts with it to go do whatever chores need to be done. Also the central storage facility for all finished goods, which are given out freely to whoever needs them or forwarded to storages to be passed out there.

Kobold Witchhut. A kobold hut for crafting magic or alchemy related things if given specific materials.

Human. Humans are versatile, starting out with no particular strengths but can learn skills that allow them to fit into any role depending on the items they're given. New tasks get implanted into them by bringing required materials to a relevant building, along with better combat stats depending on the type of training. Currently in alpha status, as they use the same buildings as kobolds, but confirmed to behave as intended.

Torchstand. A constructible torch stand that constantly emits light. Unlike regular torches, lasts forever until knocked down.

Bat. A small wandering creature that drops a disproportionate amount of tasty meat when killed. If attacked, will be infected with Fear and run away from anything hostile-looking at high speeds.

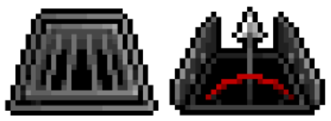
Batcave. A nest of bats, constantly spawning new ones if there aren't any nearby.

Chest. A goodie bag filled with trinkets and baubles. Can sometimes be locked, but a smart player will try to break it open.

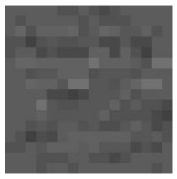
Trapped Chest. A seemingly innocent chest that activates a nearby mechanism when opened.

Door . A door that can be opened or closed. Can start out locked, but usually can be opened just by busting it down.

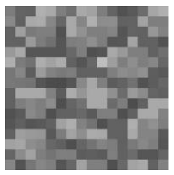
Lever. A togglable lever that activates or deactivates nearby mechanism, typically opening a door or resetting a trap or such.



Trap. A pressure plate booby trap that usually activates when stepped on, damaging the victim. Can be discovered and disabled with true sight, or by other means.

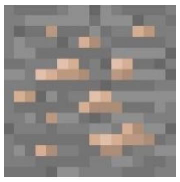


Floorwall. Untargetable walls that look like darkened floors on the map boundary, made to mark an entrance point. Used when we want to suggest the player came from a certain direction, but don't want them heading back there.



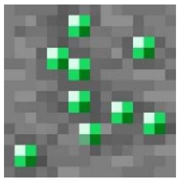
Wall. A mineable wall that doesn't drop anything if destroyed. All walls have a huge amount of defense, resistance and negation, resulting in only the 'pure' damage type found on picks or spells capable of doing any harm.

Hardwall. A wall so infused with the demonic environment that it's hardened beyond the point of being mineable. Can't be damaged in any way.



Ore Wall. A wall filled with ore, dropping some on the ground when it gets destroyed.

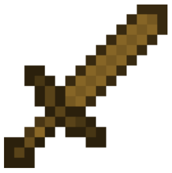
Rich Ore Wall. A wall so full of ores that it drops one every time its attacked, landing in a nearby square.



Eternal Ore Wall. A very valuable wall that sits on a surging vein of ore. Produces ores when struck a few times and constantly regenerates itself, potentially making an infinite amount. Slower to mine than normal walls, so miners will prefer to seek out other types of walls before grinding away at these ones.

Wood Rubble. Wooden debris that blocks paths. Only the player will attack it to clear the way, as everyone else will just try to path around it.

Appendix I: Item list



Copper Sword. A standard sword used for combat.

Iron Sword. A sword made of iron. Does more damage than a copper sword.

Wood Shield. A simple shield that helps defend against physical attacks.

Iron Shield. A solid shield that blocks even more physical damage.

Health Vial. A health potion that can be drunk to heal minor wounds. Anyone holding one will know to use it when low on health. Throwing it at someone slightly heals the target.

Health Potion. A health potion that heals for a large amount when drunk. Anyone holding one knows to use it when low on health. Throwing it at someone slightly heals the target.

Poison Vial. A vial of poison that shouldn't be drunk. Infects the target with poison when thrown.

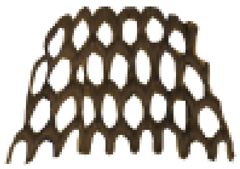
Regeneration Vial. Causes the drinker to heal over time when drunk. Less useful than simply healing the flat amount, but more cost efficient to make.

Sharesight Talisman. Allows the user to see the area around all other talismans, but has a limited number of uses before being destroyed. Useful for checking up on the base when one is out adventuring.

Flight Amulet. Passively gives the holder the ability to fly, allowing more freedom with movement and the ability to move into most occupied squares, depending on who's there.

Lifesaver Talisman. Passively protects the holder from a deathblow: if they get hit and are about to die, the talisman heals them to one HP and is destroyed.

Moonstone Amulet. Teaches the user how to use a basic magic spell with a long cooldown for as long as its held.



Hunting Net. A net that can be thrown at a small-sized target to root them in place. Mainly used by hunters to ensnare their bat prey, but works on other small creatures as well.

Ore. Rubble from a wall that contains trace elements of iron. Glows slightly. Can be smelted down to bars or used in construction.

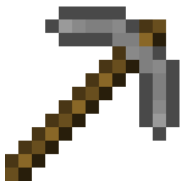
Gemstones. Some walls will drop valuable gems instead of ores when destroyed, but they're more likely found in ancient ruins. Used to create magical items.

Iron Bar. An ore refined into a piece of metal that can be used for construction or crafting.

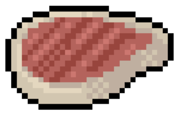
Wood bow. A common bow, capable of attacking things from a long range. Comes with an infinite amount of arrows that appear out of nowhere. Don't worry too much about it.

Iron Bow. A bow made of iron. Can shoot farther than a normal bow. Does more damage as well, so it must shoot the arrows harder too.

Copper Pick. A standard pick used for mining. Its attack damage type is pure, to break through wall defenses.

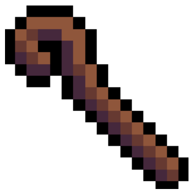


Iron Pick. A pick made of iron. Does slightly more damage than a normal pick, and allows the owner to attack walls much faster by lowering their normal post-turn cooldown by 30%.



Meat. Chunks of meat from a critter, most likely a bat. Is edible right off the ground, but preferably used in crafting. If left unstored long enough, will degrade to rotten meat.

Rotten Meat. Expired meat that causes damage if eaten. Can still be used for crafting poisons.



Staff. A piece of wood shaped into a staff. Can be used by magic users for self defense if need be.

Shaman Staff. A magical staff used by tribal magic users. Teaches its wielder how to cast a protective ward.

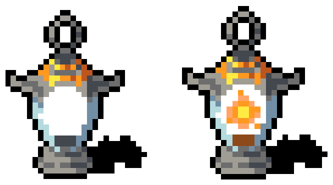
Fire Staff. A magical staff used by sophisticated magic users. Teaches its wielder how to cast fireball.

Moon Staff. A magical staff used by ancient magic users. Teaches its wielder how to cast a temporary shield on an ally (Or enemy, if one so pleases).

SORD. A cheat-weapon made for testing and debugging purposes. Has absurd attack damage, high range, and brightens up everything nearby.

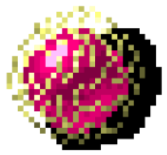
Throwing Knife. A small dagger made for throwing. Low damage if equipped, but can be thrown for a good amount and to possibly embed in the target.

Torch. A couple rags on a stick. Can be lit to provide some light, but will eventually burn out and reduce in brightness over time. If thrown at a target will ignite them briefly.



Lantern. Can be turned on to become a light source, but can also be turned off unlike many other light sources.

Telescope. A rare newfangled technology, not made for use in heavily-foggy demonic caves. Passively increases the holder's light-radius (which means they're just constantly looking through the hourglass at any given point in time, I guess).



Orb of True Sight. A mystical orb that enhances one's senses. Passively grants true sight to the holde, allowing them to see invisible people or things.

Appendix J: Status list

Faded. A spider's technique of walking with the fog, carrying it with them even when they step into the light. The target becomes invisible, but any actions or being bumped into will cause it to wear off immediately.

Poisoned. The target was injected with some sort of poison, taking constant damage over time.

Numbing. A hunter spider's poison, making the target eventually go limp if they move around too much. One can be trained to recognize the effects of the poison and stand still to avoid the effects, but otherwise any movement will increase the duration to the point they become Numb.

Numb. Once a Numbing effects is high enough in duration the target can't move at all and becomes helpless, becoming easy prey for anything nearby. Movement still increases the duration, potentially lasting forever.

Web Carried. Signifies the target is trapped in a spider's net and being hauled along behind them. If the victim stands still they'll follow the movements of the spider carrying them, but any attacks on the carrier or movement from the victim will break the binds and free them.

Scared. The target becomes scared, making them run away from any threats at high speed. Not always a bad thing, as it makes one move faster than normal at the cost of being uncontrollable.

Terrified. The target is made to be hysterically afraid, running away from all known enemies or the closest one they remember nearby.

Winded. The target just got the stuffing knocked out of them, becoming stunned and unable to act for a short time.

Cowardly. The target becomes disheartened, increasing the priority of cowardly actions like fleeing while disabling any combat-related ones.

Greed. Makes the target extremely greedy, suicidally running for any nearby valuables or stealing them from allies regardless of any other dangers.

Moral Boost. The target is encouraged to fight to the end, boosting their melee damage and increasing the priority of combat-related tasks.

Regeneration. The target heals a constant amount of HP over time.

Holy Flames. The target is immolated by holy flames, slightly burning them over time and revealing them for the duration. Also causes the target to glow brightly, allowing them to be easily tracked.

Ensnared. The target is entangled in a net, rooting them to the ground.

Warded. The target is protected by shaman magics, shielding them from bad mojo. Rejects a single debuff from effecting the target before wearing off, and has an incredibly high duration.

Blinded. The target has limited vision in some way, sometimes becoming completely blind and unable to interact with anything around them.

Fighter Training. A trait earned through training at a barracks. Makes the target stronger, healthier, able to wield most common brute-force weapons, and able to use a minor stunning blow ability.

Magic Training. A trait earned through training at a Witch Hut. Makes the target learn a basic magic spell, and grants the ability to wield and use magic staffs.

Scripted. Various level-specific effects were added that follow a unit or units, waiting for them to complete certain objectives before causing new events to happen. Usually stage based, IE: starting at 0 and causing different things as new milestones are achieved. Can range from starting dialogue, spawning / removing units / items, revealing portions of the map and more.