WORCESTER POLYTECHNICAL INSTITUTE

isoChronal Panic!

A Time Paradox in the Palm of Your Hand

Submitted By:
Edward Golden (IMGD Art)
Jonathan Saunders (IMGD Art)
Ethan White (IMGD Technical)

Advised By:
Professor Joseph Farbrook (IMGD Art)

10/11/2012

Abstract

isoChronal Panic! is a student made game developed for the browser and mobile devices. The core mechanic of the game comes in the form of “time copies” that follow the player’s previous movements, and force the player to consider their past and future actions in-game. What makes *isoChronal Panic!* unique in the realm of mobile gaming is that the game is not an app and can be played fully within a mobile device’s browser.

Table of Contents

Contents

Abstract 1
Table of Contents 2
Introduction 3
Game Design 4
- Early Vision 4
- Designing Mechanics 7
- Game Narrative 11
- Level Design 13
- Playtesting 14

Technical Design 15
- Code Structure 15
- Primary Game Mechanic 16
- Programming for Mobile Devices 17
- Sound 18
- Problems and Solutions 19

Art Design 22
- The World of isoChronal Panic! 22
- Artistic Style 22
- The Art Pipeline 26

Post Mortem 28
- Edward Golden 28
- Jonathan Saunders 30
- Ethan White 31

Introduction

isoChronal Panic! is an independent game in the truest sense of the word. It is a game developed outside of any industry framework, with original mechanics, artwork, and code design. In order to bring the game to the widest audience while still working outside the traditional game development system the *isoChronal Panic!* team has had to innovate, breaking new ground in mobile gaming, by offering a full game that can be played on any smartphone internet browser, thus avoiding the constraints of the mobile clients app policy. This new browser based design imposes several challenges to the design team, art and assets will have to be small, code needs to be organized as efficiently as possible, and every effort must be made to maximize the amount of memory available for storing game data.

While this new means of delivering game content to mobile players is one of the biggest parts of the *isoChronal Panic!* project, the new delivery pipeline would be useless to us if we don't have a fun game to deliver. The freewheeling philosophy that sprang from the rejection of the usual channels for the mobile market place has lead us down interesting paths in regards to the style and design of the game itself. *isoChronal Panic!* does something new by providing a new kind of challenge to the player. The challenge in the game comes not from an external force, as the player and his time-duplicates will be the only characters present on the screen at any time, nor does the challenge arise from the environment, as the obstacles on screen are comparatively trivial to overcome. Rather the real challenge in the game comes from recorded versions of the players previous attempts to clear the level who can interfere with their ability to complete the games objective, and forcing the player to think ahead to win the level. Introducing challenge into the game in this way provides several ways for the player to solve any given situation. For example, the player could think ahead and come up with a plan to accomplish his goal by coordinating his past selves toward his goal, or he can compete against their previous copies, having to be slightly more efficient every time.

It is important that the independent, student-made production method has comes across in the game. The blitz of having to produce the game in a scant twenty-one weeks was probably the inspiration behind the fast paced chaos evident in the gameplay. The flair for the independent can be seen throughout, whether it is the revolutionary new distribution method, the interesting mechanics, or the off-kilter art style. *isoChronal Panic!* is a game that can only have been produced by an independent student team, and it is better for it.

Game Design

Early Vision

While the main idea of having the player iterate themselves throughout the game has stayed constant, isochronal Panic! has seen a significant amount of change over the course of the project. During the initial pitch meeting we began to envision a town, which at first would be devoid of any life. The gameplay would revolve around the player populating the town with different versions of their past selves. The game has the player making a quick series of decisions on where his or her character will go and what they will do, and then start over again. Each of these quick play sessions are called “hops.” One hop would involve the player going to various places and interacting with set pieces (or not), and on the next, a time-copy would carry out those same actions allowing the players present iteration to interact with their past selves. Having thought out this game to the point where we felt comfortable putting it forward as its own game, we gave it the working title 'Ghost Town.'

It’s inevitable that a game idea goes through some reconfiguration during the design process, and Ghost Town is no exception. Speculation on how we want past versions of the player to interact with one another has lead us to the decision that we want the player and their time-clones to be able to hand items off between each other. For example, the player could place an item somewhere in the Ghost Town, and in a future hop they could return and (assuming their past version had already deposited the item) pick it up and use it. Or, the player could go to a specific location and press the button to pick up an item without an item actually being present, so on a later hop, that time clone will pick up whatever the player gives them on a future hop.

This gives rise to the question: what if it was possible to kill one of the time-clones? Also even more interesting is the question of what a game about having to outwit or eliminate your past selves would be like. Discussing this we realize that a game in which the player had to outwit previous versions of themselves is too interesting a concept to pass up, and so for now, we must move away from the idea of Ghost Town. Ghost Town would make for an interesting experience; the idea lends itself better to art and creativity, and suggests all kinds of experimental gameplay. Ghost Town will have to wait until we have more time to approach the project with the time and resources it deserves.

Moving away from the original concept of Ghost Town, we still have to figure out how we can structure the game to fit this new concept. For now, all we know is that the core mechanic of “hops” from Ghost Town is coming along for the ride and that this time the player would have to contend with a previous version of themselves moving throughout the level. In this version the player might either avoid or destroy their time clones. To this end, the *isoChronal Panic!* design team must formulate ideas as to how time travel “works” in terms of the game. Unfortunately, all of our ideas about time travel are contradictory to one another. Jon thinks that the fun in time travel is interacting directly with time copies and the environment to reach some sort of goal. Ed thinks that having all these time-clones in one place will cause a paradox unless the copies are eliminated from the time stream somehow. Ethan feels the time-clones will only cause a paradox if a past copy of the player touches the present iteration of the player.

Bzzt! - This game was put forward by Jon. The basic premise is that the player is a scientist working on a time machine when it malfunctions and drags him into a time loop. If the scientist doesn’t work quickly to fix the time machine it will malfunction again, causing the cycle to repeat forever. Unfortunately the only tool he has on hand to fix the time machine, the Time Taser, also causes localized time paradoxes and will send him back a few seconds in time to see himself doing what he just did. In this game the player will use the Time Taser to either activate an electrical circuit to open the portal that allows access to the next level, or slow down previous copies of himself. The hitch in this scenario is that the time portal only stays open for a little while after the circuit is tripped by the Time Taser, and the player must activate a circuit on the first hop. The portal is always farther away than the circuit, so the player will have to use their later copies to slow the original copy down so that the last copy can make it to the portal while it's powered up. This version of the game will require a lot of planning ahead and there were a lot of possibilities for additional mechanics like adding multiple circuits that needed to be tripped, or having automated doors close when the circuit is tripped.

This variation has a lot of potential, especially the idea of little tricks and traps we could layout for the player, and having different copies of the player interacting with one another. The biggest flaw the rest of us see with this design is that with each added obstacle or trick in design, the number of copies the player will need to complete a level would just about double. Unless the new mechanics are very simple for the player to achieve, the game runs the risk of becoming too confusing to keep track of. While a certain amount of confusion is fun, at a certain point the confusion might cross the line into overwhelming. With mechanics that demand an exacting play style, the game might not be viable for what we have to work with. There are good elements to it (having different copies of the player interacting with one another is a lot of fun) but when we hit a point of diminishing returns the game slows way down. We all agree that the important thing to take away from Bzzt is that the player and the time-clones should interact in some way.

Shoot Gun –Ed's idea was to eliminate past versions of the player in order to avoid paradoxes. Basically each "hop" would give the player a certain amount of time to head towards a time portal at the other end of the room, although the player doesn't have enough time to be able to make it all the way across the room in one hop. On the next hop, the player starts where they left off at the end of the previous hop, so that eventually they can make it across the room in a matter of a few hops. The trick is, the game remembers the player's movements from the previous hop and has a copy repeat the actions from the previous hop, and the player must eliminate these paradox copies before they can leave through the portal

To get rid of the time-clones, the player uses a "Time Ray" that removes time copies from the time stream. Basically the Time Ray shoots time particles that send time copies 5 seconds into the past where they belong. Each time-ray is only good for one shot, so unless the player can line up all of their past copies in a row, they will have to rely on shots they fired during previous hops to clear out the level. Thus, using the time ray is not so much about where to shoot, but when to shoot. This way the player will also have to avoid shots from their previous selves as they work their way across the map to line up shots on their time-clones.

Shoot Gun shows that having to run a gauntlet of past selves is a lot of fun, but the mechanisms needed to implement the game seem to require a large number of copies for each level. There is also a bit of a philosophical problem with this idea, namely Zeno's Paradox. In Zeno's Paradox, named after the Greek Philosopher Zeno, it is postulated that if two objects are in constant motion along the same trajectory, and one object is farther along the trajectory than the second object, then the second object can never catch up with the first one regardless of whether or not the second object is moving faster than the first. The explanation given for this untrue conclusion, (it's clearly possible for a fast runner to overtake someone who has a head start), is that at some point during the faster trailing objects trajectory the second object will be roughly halfway between its previous position and the first objects position, and during the time it takes to get there the first object has moved some distance. From this halfway point the second object would take some amount of time to move halfway between the previous halfway marker and the new position of the first object, during which point the first object will have moved along farther. Since it's possible to introduce an infinite amount of midpoints between the first object and the second object before the second object catches up to the first, it would seem that the first object should move an infinite amount of distance in the time it takes the second object to move through all those halfway points. With Shoot Gun we are experiencing the same problem, but in reverse. If the player is given relatively the same amount of time for each hop to get into their new position and shoot their past selves, they will have to shoot pretty early in their first few hops in order for future versions of themselves to have time to shoot them later on. In essence, the first hop will be spent getting into a good space to be shot at as soon as possible, then the second hop will be spent getting in position to shoot the previous copy. The way the math works out, the player will be gaining more and more time to shoot at their past selves at an exponential rate with each new hop in a level. This will make the games difficulty curve go down as the player reaches the end of a level, which is exactly the time they should be feeling the most tension. Another problem with this solution is that the only way to win each level would be to start at the beginning and work forward, instead of allowing the player to experiment and scramble to hit their previous versions.

Tag – Tag was Ethan's idea for how we should implement our time travel mechanics. The objective of any given level is simply to enter and go through a portal. However on the next hop, the player will have to touch a time copy before they can go through the portal again. Eventually the player builds up copies to the point where the other copies are getting in his way, and touching a particular time-clone while avoiding all of the others will prove to be a real challenge. This set-up forces the player to be mindful of all their previous copies while they hunt down one copy in particular.

The benefit of tag is that it is quite simple compared to the other prototypes. The objective is clear, and the player has the built in fun of having to dodge all of their previous selves to get to their objective. There doesn't seem to be anything wrong with this premise except we're at loose ends trying to relate to the player why they have to tag a past version of themselves. Looking at the issues in terms of design, it's easier to make up a story to reflect game mechanics than it is to make mechanics that reflect a story, so the logical next step is to move forward with Tag and sort out the story later.

All three of these game write ups would likely change drastically during the project, but of the three of possible options, the Tag variant is the most viable. We decide to go forward with Tag and make any changes later if they seem necessary after playing working with it. The important thing is to keep in mind the good and bad qualities of all three proposals, so that if we need to change our design later, the design notes from these three games will give us a place to start.

Designing Mechanics

With the Tag variant providing a firm base, we begin to design a game around hunting down one particular copy of the past player while seeking to avoid the others. Before we can get much farther, we need to sort out a mechanical issue. Specifically, how are we going to mark the copy the player needs to tag so that it stands out from the other copies? Tossing around ideas, a suggestion to just have numbers floating over the copies heads isn't going to work for us because they would be impossible to read at mobile gaming resolutions, or be made so big as to clutter up the screen. At the same time we are struggling to explain why the player needs to tag the designated time copy from a narrative point of view. One possible way to resolve both of these issues is to introduce some object the player would carry that could be passed from copy to player and then back if the player isn't careful.

Keeping in mind the ontological paradoxes involved in time travel, we have to wonder where exactly this object being traded amongst the copies comes from. What if the object just starts somewhere in the level for the player to pick up? This will give the player something to do in the first hop of the level, and give a narrative explanation for why he has to tag his time copy. When introducing a new mechanic, the first step in making sure it works is to try and break it. The obvious way to break this mechanic is to have the second copy get to the object first. Also it's possible with multiple copies to tag the first copy before the copy that would have given him the item, ever gave it to the copy in the first place. We could fix this by just not allowing the player to take it unless one of the time-clones takes it first, but maybe there is a better way to achieve this effect. In trying to test different solutions to the problem with Tag, we found something very interesting: it's a lot of fun racing around to pick up the object before your other time-clones. Racing out in front of a past self to scoop up the object first was just as fun as dodging past selves after tagging the target. While testing the solutions to this flaw, we were having more fun catching our past selves before the item was taken from them than playing the game we originally designed.

This new discovery happened to coincide with a development on the narrative end of *isoChronal Panic!*'s design. We decided to go forward with the assumption that the main character was a thief (see Game Narrative below). So our in-game explanation for why the player lost the object after being tagged by one of the time-copies is that it is stolen from him by the other time copies. The obvious question then becomes, "why can't they steal the item back?" This question, combined with the fact that it's a lot of fun to steal the object before the past version can get to it, led us to conclude that the objective shouldn't be tagging a time copy after picking up the object. Instead the objective should be grabbing the item and getting out of the level by any means necessary.

So now that we knew roughly what we wanted our game to be, we needed to design the individual core mechanics for it. This required breaking the game down into three discrete parts: getting the item, stealing the item from copies, and getting to the exit. For each mechanic we wrote a quick explanation of the mechanic itself and how it interacts with the other mechanics. For the sake of brevity, here is an edited list of our efforts:

- **Getting the Item:** The item will be placed at a fix position in the level. If the player's collision box overlaps with it, its position will track to the player's position. If a time copy's collision box overlaps the item, then the position will track to the copy's position.
- **Stealing the Item:** If a player or time copy has picked up the item, then if any other player or time-copy enters their collision box, the item will now track to the position of the colliding copy or player.
- **Exiting the Level:** The player can exit the level by colliding with the Time Portal object. The Time Portal will not record a player colliding with it unless it has possession of the item. If the item is stolen from the player, then the player cannot exit via the portal. If one of the time copies collides with the portal while in possession of the item, then the level resets to the previous hop.

From a technical standpoint, we began to make of list of all the functions that directly affect gameplay that can be changed on a per level basis. From the list we see how we can change these functions to add new challenges to gameplay. Then to bring it all together, we check how changing these mechanics will impact the three core game mechanics. These are the functions we can change on a per level basis:

1. Place the level background.
2. Add obstacles to the level.
3. Add the item.
4. Add the portal.
5. Add the player.
6. Set the number of copies needed to advance to the next level.
7. Set a time limit for the level.

By comparing the functions to the core mechanics we came up with sub mechanics. Here are the results of our brainstorming:

1. Place the level background.
    a. **Suggestion:** Have the background shift to an earlier or later time era to give players a sense of time being destabilized.
        i. **Response:** Doesn't really add a lot to game play, although we could use this to introduce new levels.
2. Add obstacles to the level.
    a. **Suggestion:** Have the obstacles move around over time.
        i. **Response:** Doesn't really make sense in game, doesn't add much in the way of fun gameplay.
    b. **Suggestion:** Add new obstacles to the stage between hops.
        i. **Response:** This seems to be one of the trickier mechanics, save it for one of the last levels.
    c. **Suggestion:** New obstacles warp in over time in the level.

            i. **Response:** So long as they reset between hops, seems like a good idea.

3. Add the Item.
    a. **Suggestion:** The Item could move over time.
        i. **Response:** This way the Item might run into past copies of the player before they actually grab it. This would also increase the time between the player getting the Item and their running for the portal, which is the best part of the game.
    b. **Suggestion:** Add multiple Items to the level.
        i. **Response:** Might be hard to parse who would win in the event that a past version and the current version bring the different Items to the same portal. Would cause hesitation, in grabbing item.
4. Add the portal.
    a. **Suggestion:** Have the portal move to a different location between hops.
        i. **Response:** Might be confusing as Time-Clones could disappear into thin air with the Item and cause the player to lose the level.
    b. **Suggestion:** Have the portal move over time.
        i. **Response:** So long as the path is the same between hops, this will solve the problem of Time-Clones disappearing into thin air. Also adds interesting challenge of avoiding the portal if player wants to give their future selves more time.
    c. **Suggestion:** Add multiple portals.
        i. **Response:** Could add interesting strategy, and maybe discourage lurking near the exit portal if the player goes for different portals at different time.
5. Add the player.
    a. **Suggestion:** Change the starting location of the player between hops.
        i. **Response:** This will really make the player hustle in later levels, but might confuse them unless we really make the player avatar stand out from the time-clones.
6. Set a time limit for the level.
    a. **Suggestion:** Increase or decrease the time limit to adjust difficulty.
        i. **Response:** Seems like a no brainer, we need a time limit to keep players from just waiting themselves out.
7. Set the number of copies needed to advance to the next level.
    a. **Suggestion:** Increase or decrease the number of copies to adjust difficulty level.
        i. **Response:** Obvious mechanic, but increasing number of copies adds a lot more difficulty than decreasing the time limit, so be careful as to how finely difficulty is adjusted.

These sub-mechanics will make up our “bag of tricks” for the game. Whenever the game enters a new phase we can introduce a new trick to make the game more challenging. Still, deciding to change how the Item functioned isn’t going to do us many favors gameplay-wise as the players need some consistent element throughout the whole game. Also messing with the background won’t really add anything to gameplay, but we can have the background change from time to time to explore our game world a bit more. Our final list of tricks looks like this:

1. Add obstacles to the level
2. Increase the number of copies required per level
3. Introduce multiple portals per level
4. Have the portals move within the level over time.
5. Change the starting position of the player
6. Warp obstacles into the level over time
7. Warp obstacles into the level between hops.

Now all that remains is explaining these mechanics in such a way that makes sense within the world of the game.

Game Narrative

With the basics of *isoChronal Panic!'s* plot in place, specifically that the games mechanics are based around time manipulation, we needed to work time travel into the plot somewhere. Before getting too far into that, we needed to answer larger questions such as: who is our protagonist? What is his/her goal and how will our players relate to that goal? We have a few guidelines going in to our story writing process; we want to stay away from the usual cadre of grim sad sacks, or squeaky-clean hero types that normally headline games these days. The game is meant to show off how fun time travel can be. With time travel, it's okay to be a little bit reckless because it's possible to go back and fix mistakes.

The main allure of time travel is that of being able to break the rules. Forget getting to peak at Christmas presents early, why not just peak at Christmas instead? Even if someone caught on to any wrongdoing, a time traveler could easily put themselves beyond the reach of justice, either by traveling to some distant point in time, or simply go back in time and undo whatever it was that got them into trouble.

As mentioned earlier we decided that the main character should be a time traveling thief. He would need a name, and a popular candidate amongst the design team was Dan Guymore, a name that's a bit of a word salad that came about when everybody on the team yelled out possible names at the same time, and this inexplicably made sense as a name. While Dan's name is an accident, the idea of portraying him as a time traveling thief is far more deliberate. Presenting Dan Guymore as a thief encourages the player to try and see things from Dan's point of view. Thieves are acquisitive and by definition steal things, which highlights our core mechanic of retrieving an object from the level. As a criminal Dan is able to sneer at laws concerning theft of property, and as such it's not out of character to have him turn up his nose to the laws of physics. The mindset of a thief on the job is to take what he came for and then get away as quickly as proper caution allows, and that's exactly how we want our players to approach any given level of the game.

There are still a few issues we needed to work out for our story. For instance: How exactly does time travel work within the world of the game? What exactly do the players steal? Why are the players going to these particular eras to steal this particular object? A tidy and economical answer presents itself in the form of a glowing blue rock we just made up called Timetanium. In the game world, Timetanium is a wonder element that allows for time travel. Unfortunately there is only one chunk of Timetanium in existence, forged into being at the dawn of time when the universe was first created. Because Timetanium offers a lot of value to a would-be thief (it's extremely scarce) finding a buyer for this merchandise would be a breeze. The story of Timetanium also suggests itself through the mechanic of the time portals only activating when the player has the Timetanium. After all, why tell the player that Timetanium is necessary for time travel, when we can demonstrate this concept through gameplay?

By adding Timetanium into the narrative, we've given the player the goal of stealing as much Timetanium as possible. By presenting Dan Guymore as a thief, the players themselves have a motivation to pursue that goal; the thrill of pulling off the perfect heist. Filling in the little details of the game is all that remained to do. From our brain storming session on mechanics, we had the idea of changing the background and obstacle images to show the game changing between different historical eras. While most time travel fiction uses the ability to play around in time to go on a tour of history's greatest hits, we wanted a bit more structure than that. The transitions into new eras needs to suggest something about the world the game takes place in because there isn't much room in the game for exposition outside of gameplay. Switching between different eras in history is the perfect opportunity to better explain the logic of time travel in game. We can show a lot with just how the transitions are laid out. For example, when the player changes eras, they only ever go backwards in time. The player can reason that this is because if they had already stolen the Timetanium in the past, it wouldn't be there in the relative present to steal again. Basically Dan Guymore will be stealing the Timetanium once in the present and then again at several other eras farther back in time. This way the particulars of time travel make sense in the context of our story within a level about the player trying to steal the Timetanium before their past selves can get it.

One of the most important aspects of a narrative is conflict. In many games, the bad guy is some sort of "other," which often means a creature or life form that's abstracted into being little more than a token representation of "bad." In most games it's implied that it is alright to kill these others, because they aren't like the player. Sometimes games do a good job of humanizing the antagonists, giving them good reasons to do what they're doing and showing the conflict in terms of two groups caught on opposite sides of an issue, where no one is at fault. That wasn't really going to work for us though. Making a browser based game for hand-held devices leaves precious little time to dedicate to story and character development.

Due to having so much fun playing around with time travel mechanics, we decided to really emphasize the cool gameplay mechanics over the story. We just would not have the time to spend getting into the backstory of the enemies. However, we didn't really feel like just making the antagonists the generic "others" as discussed earlier. While it's possible to make a game where the only conflict stems from the environment, that sort of thing has been tried before. Also in the fiction of the game world, the player has all of time and space at their command, (well maybe not space, but definitely time), so why limit a player who could, according to the fiction of the game, just pick when and where to strike? Why shouldn't the player just wait until the spikes in the pit rust over? Why not wait until those pesky walls surrounding the objective crumble to dust before stealing the prize?

A breakthrough came as we talked about exactly how players interact with their past selves. Through several of "what if" scenarios, like "what if on this hop the player gets to someplace before their past version had pressed a button to let them through?" or, "what if they somehow kill a past version of themselves?" someone out and said, "Man, dealing with pasts selves can sure get annoying," and that's all the inspiration we needed to design our conflict around. Instead of giving players past versions of themselves as resources that are difficult for them to exploit, we instead exploit the fact that it's difficult for the player to foresee the results of their past actions and so force them to deal with the consequences of not thinking ahead.

Using past versions of the player as antagonists also gives us an interesting opportunity to say something about game antagonists in general. The Other, as used by most videogames, becomes a way to pin the blame on someone else. Events within the game require the players intervention because of the actions of some outside force implying that player's problems are someone else's fault. In *isoChronal Panic!,* the enemy is the player themselves. The player is going to have to take responsibility for their own actions and acknowledge when they've made a mistake in order to fix it. The only antidote for the Other is the Self.

To sum it all up, here is the game synopsis from the About section of the *isoChronal Panic!* Website:

> *The game tells the story of Dan Guymore, time traveling Cockney Chrono-Thug, on his quest to steal the rare mineral TimeTanium from various locations in the past. Only one piece of Timetanium exists on earth, but that won't stop Dan from stealing as much as he can throughout time. And since he has the only time traveling Chrono-Bowler Hat in existence, the only thing that can get in his way is himself.*

Level Design

After we had our mechanics and narrative firmly in hand, we set about making levels. What we wanted to do is have each level be deceptively easy the first time through, but then become a lot harder with the addition of extra copies. For example, we designed a level that is just a straight shot between the player, the Timetanium, and the portal. If the player does the obvious thing and heads directly through the Timetanium into the portal, he'll never be able to catch himself on the next hop. These are the kinds of levels we wanted to produce; levels that take thought to overcome.

To get an idea where to begin with our level design we needed to look at the bag of tricks we came up with while we designed our game mechanics:

1. Add obstacles to the level
2. Increase the number of copies required per level
3. Introduce multiple portals per level
4. Have the portals move within the level over time.
5. Change the starting position of the player
6. Warp obstacles into the level over time
7. Warp obstacles into the level between hops.

Also, we wanted to divide different sections of gameplay between different time eras to give each era a distinctive feel. We really only had time to generate assets for four different time eras, so we would have to divvy up all of our tricks between the four different eras.

With our goal being to have the game get harder as it goes, we've added in each sub-mechanic in order of how difficult it makes the gameplay, with mechanics that complement each other or that add about the same level of difficulty paired off into each era. Here is our final result approximation of which trick to add in a given era:

- **Era 1:** Present Era
  - Add obstacles to the level

    - Increase the number of copies required per level
- **Era 2:** The 60's
    - Have the portals move within the level over time
    - Introduce multiple portals per level
- **Era 3:** Medieval Times
    - Change the starting position of the player
- **Era 4:** 1,000,000 BC
    - Warp obstacles into the level between hops
    - Warp obstacles into the level over time

This list guarantees we can add new and interesting concepts to our levels throughout the entire progression of the game. Applying our macro level philosophy and to each level individually, we can arrange the game so that each level can introduce a new situation or way of thinking to the player. While we don't necessarily have to introduce a new mechanic on each level, we must introduce a new challenge every time the player moves forward within the game to keep their interest.

Playtesting

With our basic mechanics and a way to quickly create new levels in hand, we would just need a way to determine that the levels we create are fun for the player. To make sure the game is fun, we would need to exhaustively test the game on our own, but sometimes a designer is just too close to the game to notice a bug. We would need playtesters to come from outside of our design team and provide enough feedback to point us toward what we need to focus on when designing the final version of *isoChronal Panic!*

While playtesting is important, it's not some sort of crystal ball that holds all the answers. One of our Professors, Britt Snyder, warns against trusting playtesting implicitly. Britt tells a story of when he worked in the games industry, and how playtesting had been used as a means of hedging the design team's bets, basing pretty much every gameplay decision on what was said during the playtest. If we use playtesting merely as a way to cater to the broadest possible audience, then the game is going to end up satisfying no one. In particular, Britt warns that playtesters tend to shy away from the more out-there ideas. Professor Snyder shared with us an anecdote of a game he was working on that had a strange diversion where the player had to fight a giant tiger. The tiger didn't really have anything to do with the rest of the game, but it was one of the few challenging and engaging parts built into the game. Unfortunately the rest of the game was boring and rote, so playtesters picked out the tiger fight as odd and as a result it was removed from the game. In Britt's opinion the game needed more tiger fights, not less, and argued the game would've been a lot better if they had gotten rid of the rest of the game and made it more like the tiger fights.

The moral of the story is not that playtesters are uncultured philistines who don't know what they want, but rather playtesting is a blunt instrument for finding incongruent elements in a game. It's up the game designer to determine whether these incongruent elements need to be eliminated, or if they work for the game. *isoChronal Panic!* is a weird little game to begin with, so we would have to keep that in mind when analyzing the results of the playtest.

Playtesting is an important part of any game design process. Fine-tuning is extremely important in a twitch-based game like *isoChronal Panic!* and since we were working on a relatively new, (to the design team at least), platform we would have to make sure the game translates to touch based controls. Thus, the last portion of our development cycle was completely dedicated to testing and adjusting the game.

Our playtesting was conducted in three sections. First, we needed to make sure the basic mechanics would not only be functional, but also fun before even bringing the game to mobile devices. To get this data, playtests started with a simple browser version. The browser version of our game has a lot fewer moving parts then the mobile version and is therefore simpler to make quick adjustments as suggested by our play-testers. Browser based playtesting also has the advantage of opening the playtest to a wider audience as not everyone has access to a mobile device. We decided that we can't use the participants from first playtest for the mobile playtest even if they had access to a smartphone, because they might give false positives due to preconceptions formed during the browser playtest which would bias the results of the mobile playtest.

Technical Design

Code Structure

isoChronal Panic! Is programmed using HTML5 and Javascript, because these programing languages strike us as being fairly simple to use, and they are also accessible on most browsers and phones. HTML5 also allows us to develop for the iPhone market, without having to go through the bureaucracy of Apple's App store. While working with the language we're discovering that despite Javascript being a scripting language, it is still possible to use it as a pseudo object-oriented design language. Each 'object,' like the Timetanium, or the Player, or even any of the various game managers, have their own javascript file, called a class. From these classes we can do anything we need to with the basic game objects by making subclasses and prototyping off its 'parent.'

The games "Main" file contains all of the games global variables as well as a call to start the Game Object Manager, which strings together the rest of our game. The Game Object Manager starts up the other managers, loads the required images and sounds into the Resource Manager and Sound Manager listens for input and runs the game and render loops, respectively. It might be more appropriate to just call it the Game Manager, but the particulars of Javascript conventions means Game Object Manager is pretty much the same thing. Other than that, the bulk of the code is in the Player class which includes interpretation of input and calculation of collisions and movement, and the Level class in which each level is encoded via a switch statement.

Primary Game Mechanic

While programming *isoChronal Panic!,* one of the most important design concerns was how to generate the time copies. In the end we just used the simplest and most direct approach, which is to record the player's position in the update loop, and then have the copy follow that record. We didn't plan on using this method originally, since storing several values of each tick of the game loop for each copy seems like an inefficient a use of the memory. But when we thought about it, just storing all the data makes the most sense, as our other idea of only recording the player's destination whenever it changed ended up being too bug-prone, (see Problems and Solutions). Also, we realized that because it's possible for the player to change their destination constantly, we would need to account for the possibility of storing data every game tick regardless. Programming requires taking into account the worst case scenario, and seeing how recording the players position at every tick would be exactly equivalent to the worst case scenario, we decided we might as just record the players position all of the time as we originally intended.

When the player is initialized, the game creates three new arrays, one for x-coordinates, one for y-coordinates, and one for index numbers indicating which animation is currently in use. Then in the player's update loop we add the appropriate value to the end of each array, (the coordinates were also rounded to prevent the cost of unnecessary precision).

```
//Update the current Past Record
this.xCoords.push(Math.round(this.x));
this.yCoords.push(Math.round(this.y));
this.animations.push(this.animation);
```

Then, at the end of each hop, if the player succeeds, this information is stored in a Past Record, which consists solely of all three arrays. This past record is then added to a global list of all past records. At the start of each level, this list is used to assign a past record to each copy.

```
for (i=0; i < g_Copies.length; i++)
    {
      g_Copies[i].startupCopyPlayer(this.level, g_Records[i], i);
    }
```

Then the copies simply set their own coordinates and animations according to their past record, based on how many ticks have gone by.

```
this.x = this.record.xCoords[this.ticks];
this.y = this.record.yCoords[this.ticks];
this.updateAnimation(this.record.animations[this.ticks]);
```

Should a copy succeed before the player does, the most recent past record and copy are removed from the scene.

```
            g_Records.pop();
            g_Copies.pop();
            g_ApplicationManager.hop();
```

The code isn't very complex, but the gameplay produced as a result, is.

Programming for Mobile Devices

Now that the core mechanics of movement and copies were working, we needed to work towards getting the game working on mobile devices. The cornerstone of any smartphone game is gathering input through the touchscreen. Initially, it seemed that the standard functions for gathering mouse clicks on a normal computer would be enough. After all, that's how mobile HTML works! However we soon found out that programming for mobile platforms requires a bit more finesse than that. Trial and error has shown that this is the real way to get a touch pad to pick up touch inputs:

```
            this.canvas.addEventListener("touchstart", touchStart, false);
```

This adds an event listener to the canvas, which sits in wait until it hears the input from a finger pressing on the touchscreen. When the input is gathered, it activates a function elsewhere, named touchStart.

```
            function touchStart(event)
                {
                  //Store the coordinates of the touch
                  g_MouseX = event.touches[0].pageX;
                  g_MouseY = event.touches[0].pageY;

                  //Process the event for any objects that use it
                    for (var x = 0; x <
            g_GameObjectManager.gameObjects.length; ++x)
                    {
                        if (g_GameObjectManager.gameObjects[x].touchStart)
                        {

            g_GameObjectManager.gameObjects[x].touchStart(event);
                        }
                    }
                }
```

While this code is similar to how mouse clicks are normally gathered by a device, mobile devices won't work unless it is phrased in the grammar.

While we were at it, we decided to code another event listener for touch movement, so mobile players can actually touch and drag their character. The function to handle this is a bit more involved than the above example.

```
            function touchMove(event)
                 {
                  //Prevents the touch event from being propagated to
            the browser
            event.preventDefault();


                // get canvas position
                var obj = this.canvas;
                var top = 0;
                var left = 0;
                while (obj && obj.tagName != 'BODY') {
                    top += obj.offsetTop;
                    left += obj.offsetLeft;
                    obj = obj.offsetParent;
                }

               g_MouseX = event.changedTouches[0].pageX - left +
      window.pageXOffset;
               g_MouseY = event.changedTouches[0].pageY - top +
      window.pageYOffset;
          }
```

Each time a mobile user moves their finger on the screen, this piece of code acquires and stores the coordinates of their finger with respect to the canvas.

We wanted to make the game as polished as possible on all platforms, which meant putting in a considerable amount of manpower into getting it to look like a native application on an iPhone or Android. To do this we resized the browser window to fit a variety of mobile device sizes, as well as orienting the screen horizontally while maintaining full screen resolution.

After a bit of a wild goose chase trying to find a way to seamlessly resize the entire canvas, along with every element inside it, we finally understood that doing that would entail manually resizing each image and realigning every coordinate to get it to work properly. Resolving it this was the only way to do it. Actually doing it was fairly simple because we knew the standard screen ratio. It's quite clever really:

```
g_RatioX = (window.innerWidth / 600);
g_RatioY = (window.innerHeight / 360);
```

The first thing to do is find the ratio between the default size of the canvas, as in the frame of reference that all of our game assets and levels are based on, and the size of the screen on which the game has to conform. Once we have this, we comb over a large portion of the codebase to find any mention of the size parameters of objects or the canvas itself. Then we just multiply them by `g_RatioX` or `g_RatioY`, which resizes the object appropriately. A day's worth of debugging later and we finally caught every instance of sizing and now everything resizes reliably.

Sound

When developing for the iPhone, an important concern is how to handle sound. Apple restricts loading sounds for non-native applications, which makes a game's normal array of background music and sound effects nearly impossible to implement in a typical fashion. For instance, the iPhone only allows a single sound channel to be used at a time, rendering it impossible to play simultaneous sound effects or music. We began the project hoping to get around this restriction by devising a system of pre-recorded overlapping sounds. For instance, having one sound effect for the player picking up the Timetanium, and a separate sound for when a time copy picks up Timetanium and then combining the two tracks to let the player to create a third sound effect that plays when both actions occur in quick succession. Since there can't be two sound channels open to play both sound clips by default, we could detect if both actions occurred at once in the game and then play the overlapping sound instead of the two individual sounds. We didn't have very many sound effects planned, so we considered the work overhead for making these additional overlap sounds fairly low.

A problem with this system was soon brought to light. The overlapping sound files would, if used anytime with more than one sound overlapping, end up feeling unresponsive, as the sounds would have a hard time playing at the same time that an action occurred. In a real-time game world such as ours, actions with an associated sound effect could occur at any moment, and there would be many instances of sound effects and music overlapping each other. However, it would be fairly rare for two sounds to perfectly overlap in the same way we had pre-recorded. The two sounds could start at any moment, so the degree to which they overlap could be near infinite. We came to the realization that the only way we could have sound working on an iPhone at all would be to choose between sound effects OR background music, since the music would almost always be overlapping with a sound effect. In addition, if we went with sound effects, we would have to accept the fact that they would not be able to overlap. If a sound was triggered before the end of a previous sound, the first sound would have to be cut off. Despite the restrictions, we decided that our game would best be suited with sound effects, even if they did have to get cut off from time to time.

Since we can only have one sound channel at a time, we decided we might as well only have a single sound file as well. This cuts down on precious memory usage by playing each sound effect by tracking to different parts of one sound file with all of the sounds on it. This makes it so that we cannot accidentally play more than one sound at a time, decreases overall file size, and increases memory efficiency.

Problems and Solutions

In every game there are a lot of little bugs to fix and problems to solve over the course of a project, but most of them can be resolved in one or two coding sessions. However, there are always several bigger issues which plague the project for weeks, or even months. *isoChronal Panic!* is not immune to these problems. Specifically, we were having issues with generating copies, getting touch to work, and putting sounds in the game.

At first we thought we could get time copies to work without recording too many values. The solution seemed to be to have the player initialize with a recorder object. The recorder is supposed to keep track of two lists of variables, and each should update only when the player changes their destination. This way, the game should receive the coordinates of their new destination (the point on the screen where a click has happened), and the timestamp of when they began heading towards it. When a hop occurs, the record should be handed over to a new copy, which has a lot of the same code behind it as the player, but without the control input. The copy should then follow the steps laid out by the record exactly, by traveling in the direction specified for the amount of time specified.

```
this.recorder.recordObject(new
Coordinate().startupCoordinate(this.destX, this.destY));
```

This line in the player object is called every time the coordinates of the clicked mouse, causing this.destX and this.destY to change. A new coordinate object is then created on the spot, and sent to the recorder object that the player is holding onto. The recorder puts the new coordinate and a time-stamp into an array.

```
if (this.tick === this.record.duration[this.count]) {
 this.activateMovement(this.record.direction[this.count];  this.update
Required = true; this.count++;}
```

This little block in the update function of the copy class is what reads a completed record. Movement works the same as with the player, but instead of being controlled by coordinates via mouse input, the copy is controlled by coordinates stored in the record. The third line tells the copy which the animations it should display. The last line increments a simple count variable, so that when the time comes, the next coordinate in the array is read. This method is fairly simple and works pretty well under normal conditions. There is a major problem with it though: lag. It doesn't actually cause any lag itself, but when playing over the browser, especially on a phone, lag *will* happen. The first line of the above code is an if statement that will let the rest of the block run when the current moment in time, or tick, of the game matches the time-stamps in the record. If the game lags, then often the tick variable will skip ahead a little bit, causing a few chunks of time to go ignored. If this happens near when the code is executed, it might not get past the initial if statement. For instance, if an action is supposed to occur at tick 12 and some lag starts up at tick 10 and lasts until tick 13, then there's a good chance that ticks 11 and 12 will be skipped over entirely. The change in direction never happens, and the naughty copy just keeps walking in the same direction. This and several other lag related bugs suggested to us that we should switch to a different method of making copies. Another factor in this decision is the fact that, if the player moves by clicking and dragging their character (which many players, including ourselves, ended up doing most of the time), their destination would constantly be changing, costing us just as much computation-wise as our current method.

Getting touch to work was also taking much more time to get right than we anticipated. The problem isn't that no precedents exist for these kinds of problems; it's just that there are too many different ways to handle it, with no accepted standard. It's easy to program the game so that it starts up when touched, but getting the player character to react to touch inputs is another story. Looking at the code now, the problem seems to be that we were retrieving the coordinates from the touch event incorrectly. Unfortunately, touch events have several different variables which contain some version of the touch coordinates, depending on what they're relative to and whether they were new touches or old ones which were changed.

Art Design

The World of isoChronal Panic!

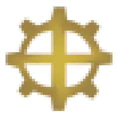
When designing the art style of *isoChronal Panic!* we already had some idea of what the world of the game would be like. The items in our game world should follow a common theme, and the best way to do that is to identify important objects and base the art design around their signature look. The essential nature of the game's themes and mechanics can be brought through in the art, but first some thinking about the world of *isoChronal Panic!* would be in order.

One of the main themes of *isoChronal Panic!* is finding multiple solutions for the same problem. When the player is forced to play through the same level again with the complication of the time-clones added into the mix, the player must come up with a new solution to the previous problem. The same line of thinking is evident in the environments of *isoChronal Panic!* In each era of play, whoever possesses the Timetanium has to solve the problem of how to store and protect it. In the present era, the Timetanium lab employs the latest in magnetic-field suspension, but in the 60's era the Timetanium is held within a custom built containment chamber that looks kind of like a diving bell, in the Medieval era, the Timetanium is embedded in a stone, and in the Stone Age the Timetanium is surrounded by a ring of skulls. The same problem is presented four different times, and is solved in four different ways, which plays into the themes presented in the games mechanics.

Themes of understanding also play throughout the game. The world's theoretical understanding of the Timetanium and time travel diminishes the farther back the player travels in time. There is a contrast between the knowledge shown in the world and the knowledge possessed by the player, the greater he understands how to manipulate time, the less the world around him seems to understand the same principles. The player feels his mastery of time grow relative to those who inhabit the environment around him, lending to a sense of accomplishment in being the most knowledgeable about time travel.

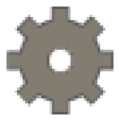
Artistic Style

Mobile devices lend themselves to a 2D art style. Because of their limited screen space and device memory there is little point to getting exceptionally detailed with the graphics. At the same time, the graphics that are put on the screen need to convey a meaning and allow the player to know what is what even at a small size. Below is an early example of a gear mechanism we were going to use for an earlier incarnation of the game:



While this graphic does resemble a gear, it could also be so many other things as well, like the wheel of a ship, a piece of wall decoration, or a water valve. Sure, if it was big enough, and enough details were added it might come across as a gear, but at the sizes needed for mobile devices, no one would really understand what they were looking at. The artistic flourish of the slight gradient in the center of the object makes the graphic pop, but it's flash over substance. The viewer would see the gradient first and the fact that it was a gear second, or possibly not at all. The design of the objects in the game must convey the objects purpose and identity right away.

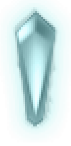
While that's easy enough to say, it's quite another thing to learn how to do. There is no magic piece of advice that will automatically give an art style that level of clarity. The best way to learn how to draw a gear is to just keep drawing gears until something looks like a gear. If we could create something simple like a gear, then we could analyze what makes it distinct and apply that to everything else. So, going back to the drawing board, the art team came back with this:



This gear rejects the flash of the previous attempt; it's all substance. While it doesn't have the same pop as the first gear, a viewer can tell what it is at first glance. Even at the small sizes required for mobile devices, the purpose and function of the object comes across. This gear represents the first great breakthrough in the art design of *isoChronal Panic!* and it would be used to turn the engines of creativity through the rest of the game. The art team quickly vivisected the style of the gear in order to develop some fundamental rules for the rest of the art style, such as:

1. **Simple Shapes-** The shape of the object should suggest its identity and function.
2. **Curves and Lines-** Contrast and visual stress in the object is primarily shown, not by contrasting colors, or varying line thickness, but by the juxtaposition of straight lines and curves.
3. **Flatness-** Objects flush with one of our primary planes will appear flat, with slight suggestion of depth around the edges.
4. **Color-** Different surfaces on the same object are denoted by different shades of the same color, unless the object would have different colored surfaces normally.

While the gear serves as our first breakthrough, it's not really a core part of our art. In the end it's just a little piece of decoration on the back of Dan Guymore's hat. To follow up on the gear, we created one of the core thematic pieces of art in the game, the Timetanium. Using our previous method of just drawing many different art assets and then sifting through them for "the one" the art team came up with this:

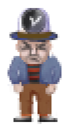


This is exactly what we wanted our glowing time crystal to look like, and thus our long search for an artistic style had come to fruition. Good thing too, because making art assets by trial and error might make for great insights, but it takes up a lot of development time. Now that we have a good foundation for our art assets, we would need to switch from creating rules based off of the art we designed, and start creating art based off of the rules we'd designed.

In order to keep the visual style of the Timetanium, all “Time Stuff” should have a greenish blue color pallet; this means things like time copies and the time portals should all be the same color blue, or feature it prominently. However, the time portal needs contrasting elements to give it a chaotic feeling, so in this case greenish blue transforms into green and blue as seen here:

The powder blue is still the primary color, but highlights of green show around the edges of the central swirl. With the Timetanium and the portal in place, *isoChronal Panic!* has two of its core elements in place, but still lacks the critical element of a player avatar.

Dan Guymore has to do a lot of things at once, he has to be engaging to the player, express the personality of the character, project a unique silhouette, match with the art rules we’ve already established for game objects, and suggest his connection to the time travel mechanics of the game. Below is the final design for the Dan Guymore sprite, which is followed by an in depth analysis of how we will achieve all the goals we set forward for our character:



So right off the bat, Dan’s engaging in his novelty, he’s a boxy cockney with a clock in his bowler hat. His facial expression gives the impression that he’s somewhat of an unpleasant character which gets at his criminal nature, while at the same time his baby-faced cheeks keep him from being totally off-putting. The Chrono-Bowler gives his whole body a unique shape that the players’ eye is instantly drawn to at the start of a level. From a purely shape based perspective he’s composed of several simple shapes, the clock is a series of intersecting ellipsis’, the head a rounded square, and the body an oval crossing the “T” of a mostly rectangular body. The curve of the bowler and his sagging shoulders create a curved top half that contrasts with the rest of his boxy frame. Dan as a whole appears somewhat flat, but the curve of the dome on the bowler and shading on his clothes suggest depth. Finally the primary blue of his jacket and hints of blue tint of the Chrono-Bowler, tie him into the color themes of time travel.

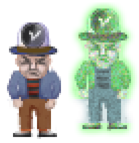
We feel that this version of Dan is perfect, but we still needed to differentiate Dan from his time-clones. Our initial plan was to just make the time-clones glow bright blue and give them some “time static,” or tiny built-in artifacts to give the impression that the copies were more than a little insubstantial. The effect is demonstrated below:

The design team loves this look for the time-clones; the color ties into our time color scheme, the static shows that there is something different about him, but it isn't intrusive enough to hide the characters features.

However there is one problem with this Dan; he's too blue. Dan looks great by himself, and the copies look great by themselves, but in playtesting, players had a hard time telling the copies apart from the player. A lot of player comments were telling us about how the character would seem to move on their own and screw up the player's strategy for the hop. There is nothing to indicate that the player is ever moving on his own, so we think that some players just could not distinguish between the player and the copies This was a problem; either take Dan Guymore back to the drawing board to make him less blue, so he's distinct from his clones, or change the color of the clones and risk removing their thematic link to time travel.

In the end we were saved by a thematic embellishment we added earlier. As mentioned above, the time portal has green highlights to contrast with the blue, and contrast between Dan and his time-clones is exactly what we needed. By making Dan green, we still have ties to time travel, by way of the shades of green already present in the time portal, and tie the color scheme in with the game mechanics as well. Dan and his clone's movement around the screen now resemble the blue and green swirl of the portal, each chasing after one another into infinity. The time portals themselves were meant to seem a little chaotic, the bands of green representing disorder in the otherwise perfect spiral of a time loop, so too do the green colored copies represent chaos injected into the otherwise simple world of Dan Guymore. Below is an example of one of the time-clones in action:

To see the contrast of the copy and Dan Guymore side by side, observe the image below:



The redesigned copy seems to be just the ticket, as we no longer receive any playtest reports about the player moving on their own.

The Art Pipeline

The art pipeline for *isoChronal Panic!* , a four step process, like most other elements of our game design is a little bit unusual. The art in our game starts out in physical media, but then is scanned into the computer to be put through the wringer of several digital art programs. This way, our art comes across as hand drawn, but at the same time exceptionally clean and crisp. The objects in the game are distinctly digital in their nature, but have warmth and a sense of crafted care because of the way our art process mixes different artistic mediums together to achieve the final product.

The first step in the art process is to sketch out the level and the various objects within it in detail. Using as many reference images as we can gather, the art team tries to create the feel of the each object and environment, while still capturing the artist's own style. The general sketch of the level is meant as more of concept sketch, and can be pretty loose and general, however the individual objects within the level are sketched in more detail. Although the sketches of the objects still serve as concept art, they're also often used as direct reference by scanning the images into the computer and tracing them in other art programs farther down the line.

The sketch of the level is also made into a computer render in the art pipeline's second step, although not as a direct sketch. Instead, the level sketch is used as a reference for a 3D model made in Autodesk Maya to give us a sense of perspective. Each item in the sketch is made into a general shape, with a little detail to get a better sense of the object's place in the room, and how it should be viewed from perspective. The game has a little bit more of a skewed and cartoony style to it, so the perspective of the objects is played with a bit, by rotating the objects slightly to create an off-kilter feel. The renders are then referenced when drawing the finalized version of the level.

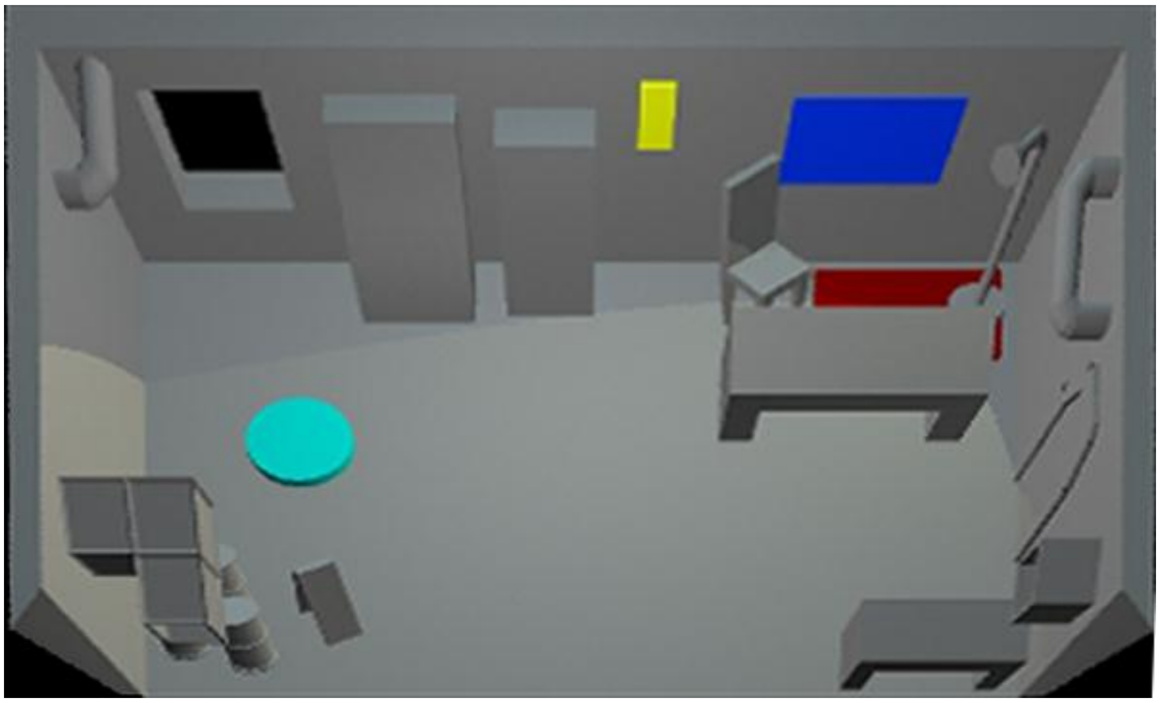


Figure 1 3D Maya mockup of first level

Now that we know where the objects should be in the room, we start the third step of the art pipeline: tracing the object sketches in Adobe Illustrator. As mentioned during the first step, the sketches were scanned to create digital images of the sketches; the sketches are then placed onto the art board in Illustrator and made into templates to trace our game art. Illustrator, as a vector art program, allows us to take our initial rough sketches and turn them into crisp and clean works of art. The rough line work of an imperfect hand quickly becomes smooth Bezier lines in Illustrator, and because Illustrator is a vector art program, we can easily scale down images to their proper size without sacrificing detail or creating compression artifacts. While the art is going through its Illustrator phase, changes from the initial sketch start to emerge. Color can quickly and easily be added, more details can be added on top of the original sketch, and most importantly, the digital framework allows the artists to experiment without risk of losing the original. Many of the objects and environments change drastically during this step of the art pipeline.

Now that the objects have been brought into the digital realm, it’s time to put the finishing touches on them. From Illustrator, the images are exported into Photoshop for final details and to get them in the right format for the game. Most objects are pretty much ready to go after briefly passing through Photoshop to format them correctly for the game, but sometimes details such as shading or weathering are added in during this step. Illustrator is too fine an instrument to add shadows or discoloring to these objects, as well as adjustments like color balance and blending. After these final adjustments, the images are ready to go into the game. As a result of this meticulous process, each object looks polished and professional.

Post Mortem

Edward Golden

Looking back on my experience with *isoChronal Panic!* I really have to say that the work required for a small student project like this was a very different beast than I thought it would be. I knew it was going to be a lot of work going in, but I was still a little surprised just how many hours it needed. But that's not really what surprised me the most. After being named Lead Artist, I initially thought that I'd be working in with the rest of the group and that would entail a lot more collaborative effort between team members, but the sheer amount of work we had on our plate for the project meant that most of the time I spent was wholly devoted to art asset creation (to clarify, as far as I know I was the only one working completely by myself most of the time. I can't speak for the other two members). It's not that we never saw each other, or looked at one another's work, it's just that all of our project roles became so specialized. We were all devoting so much time to our particular specialty that it was easier to get into the rhythm without having to constantly arrange collaborations with my other partners.

When I said we were specialized, I didn't mean to suggest that we only did things within our specialized domain, as in the artists only made art and the programmers only programmed. We were always playing around with the game and looking to see if the assets we added to the game were working right or needed adjustments. If I needed to tweak some aspect on the technical end of things to make things look right, I got wrist deep in the code and made the adjustments needed. While I'm not a code whiz by any stretch of the imagination (and I don't know what a majority of the code in the game does, despite the fact that it's pretty well commented), I know enough to do what I need to do, and what I don't know how to do, I learn how to do on the job.

As to the game itself, I was a bit worried about its playability at first, but during playtesting I learned we were on the right track. The most important thing I think we accomplished was making the game fun. We got a lot of comments during playtesting that the game was a blast to play, and that's when I knew we had something really good on our hands. Whatever else can be said about the game, we found a way to make it fun in a fast and frenetic way. What's more it is simple to pick up and play. After the player figures out the controls, they instantly grasped the games objective. Fun and simple to understand; these are two descriptions that every game should hope to receive.

Another area where I think we absolutely knocked it out of the park is the distribution pipeline. Anything within an online browser can access and play *isoChronal Panic!* Even smart phones and other touch screen devices have access, all without having to go through any kind of App Store. I think that this browser-based solution is really something revolutionary that needs to get out into the wider world. With something like the Internet, there shouldn't be any way to control or restrict how someone can access content, even on platforms like Smart Phones and Touch Pads.

That's not to say the game was developed flawlessly. There are still a lot of things that could've made *isoChronal Panic!* better. Outside of the obvious stuff of wishing there had been more time to add more content assets and levels, I think there are a few things in particular that really would have made *isoChronal Panic!* one for the history books. For one thing, I really think we could have benefitted from having another team member on board. As it stood, we had exactly one person dedicated to each facet of the project; Ethan programming the core, Jon porting it over to mobile and myself producing the art. A fourth member could've allowed us to have a bit of support and do more with our time. As it stands right now, *isoChronal Panic!* has a very solid core and it's just a matter of squeezing as much variety out of the basic concept as possible. We thought that keeping the design team lean and mean would allow us to just power through the making of the game, but looking back, I think we should've broadened our stable of talent just a little bit. Another thing I really wish we could have gotten to was adding music to the game. I mean, I don't think music is necessary to a game like *isoChronal Panic!,* but just from a technical stand point, being able to get background music into the game would've shown that these artificial restrictions placed on games on mobile devices are nothing but a complete illusion.

Jonathan Saunders

The course of this project had a lot of ups and downs for me. Coming up with the initial concept for 'Ghost Town' and fleshing the idea enough to start considering it as an MQP, teaching myself enough Javascript in just a little over two weeks to code the backbone for the entire game, brainstorming new twists or mechanics, realizing that there was a lot more cool stuff going on with this thing than I had anticipated, working closely with some good friends and figuring out the best way to get things done with them, learning how to take a project all the way from nothing but a thought to a polished project, and watching players rejoice in the miniature world that I helped forge were just some of the great experiences I had over the past few months. On the less fun side of things, we had to work through dealing with the badly-mapped hedge maze that is mobile development, code bugs that seemed like they would never get solved, creative differences on what the very goal of the game would be, and running headlong into issues that still haven't been adequately sorted out by the community as a whole.

Most of our problems seemed to stem from the mobile side of things; at least that's how it looked from my perspective. Admittedly, I'm probably biased in that respect, since I ended up becoming the sole worker behind everything mobile. Originally I intended to work 50% on code and 50% on art. Looking back, I ended up splitting up more work more like 10% art, 30% regular code, and 60% mobile code. I especially missed not being able to work more with art, and I know Ed missed that too, since he was counting on me to join him after I had made the game's framework and passed the majority of the normal code duties on to Ethan.

So now I can't help but think: where would the game have gone if we hadn't decided to go mobile? What would be different if I had been able to devote my full attentions to the game itself, rather than its platform? Surely this is an exercise in futility, but regardless, we would likely have been able to expand the functionality of the current game and add more features and more levels, all with an even higher level of polish. We would have avoided hundreds of man-hours spent researching, figuring out, and debugging things like touch controls, canvas resizing, and sound. I would have been able to continue honing my skills in art and design and programming. On the other hand, I also would have missed out on a pretty huge learning experience. I now know what it's like to work on a project with a series of legitimate roadblocks, and to get through them with an actual product on the other side. I also got my first real taste of developing for mobile devices, a skill which is only becoming more in-demand in the web and game industries. Not only that, but with a working mobile version we are able to reach an even wider audience than we thought we would. From my research I've found that companies are time and again surprised by how many people tend to gravitate towards mobile versions of their games over PC based ones. I can only hope we get to see that phenomenon first hand!

Ethan White

I still remember when, about a year ago as of writing this, Ed and Jon approached me to ask if I wanted in on their MQP idea of a game where the player played against their own past selves. The idea sounded pretty interesting, and who wouldn't jump on a chance to receive course credit for making a cool game with their friends? Of course, there was a lot to figure out: what programming language to use, what platform to make it for, what exactly the gameplay would be like, etc. For most of the project, it felt like we were progressing pretty slowly. From week to week, we would add a small feature or fix a problem that had cropped up, while putting off a larger one for later. During the summer, I worried that we wouldn't be able to finish on time, or if we did, that the game would be a rush job, made only to satisfy an academic requirement. But we solved our fundamental problems one by one, and when we looked back on what we'd accomplished after summer was over, I realized that we'd come much further than I'd thought.

I haven't played many browser/app games, but I like the way ours works. In particular, I enjoy the theme of "getting in your own way" which is present throughout the game. It's represented both in our barebones plot (in which Dan Guymore is constantly stealing from himself, and using time travel to make this profitable for at least one of him) and the gameplay (where the player constantly struggles to overcome their own best run, while also needing to make sure not to do *too* well early on). Thinking a bit more about it, our game is a lesson in empathy; while screwing someone else over may not seem like such a big deal at first, by forcing the player to act as both the screwer and the screwee, we make the player realize exactly what they're putting others through.