



# Simulator for Teacher-Student Classroom Interactions

**A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science.**

## **Project Team:**

Mary Braen ([mebraen@wpi.edu](mailto:mebraen@wpi.edu))  
Gabriel Camacho ([gcamacho@wpi.edu](mailto:gcamacho@wpi.edu))  
Emily Lin ([elin@wpi.edu](mailto:elin@wpi.edu))  
Ryan Luu ([rmluu@wpi.edu](mailto:rmluu@wpi.edu))  
Aidan Mulcahey ([ammulcahey@wpi.edu](mailto:ammulcahey@wpi.edu))  
Jonathan Palmieri ([jdpalmieri@wpi.edu](mailto:jdpalmieri@wpi.edu))

## **Project Advisor:**

Professor Jacob Whitehill

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review.

For more information about the projects program at WPI, please see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>

# **Abstract**

We developed a two dimensional graphical classroom simulator to help train teachers to perceive teacher and student interactions. Each simulation lasts up to 20 minutes and contains one teacher and multiple students. The storylines are generated randomly and employ 32 unique events and emotions, such that events are arranged in a logical sequence. Using this simulator as a training and a data collection model, our project advisor sought out to help the next generation of teachers better prepare themselves for the real world.

# Table of Contents

|   |    |
|---|----|
| <u>Chapter 1</u> : Introduction           | 3  |
| <u>Chapter 2</u> : Background             | 6  |
| <u>Chapter 3</u> : Specifications         | 9  |
| <u>Chapter 4</u> : Software Tools Choices | 15 |
| <u>Chapter 5</u> : Implementation         | 23 |
| <u>Chapter 6</u> : Conclusion             | 33 |
| References                                | 36 |
| Appendix                                  | 37 |

## Chapter 1: Introduction

Teaching as a profession has many challenges and requires constant attention to multiple tasks simultaneously, including planning lessons, managing student behavior, and monitoring student progress. However, one of the most significant challenges that teachers face is the need to divide their attention among all their students, especially with younger age groups. Research suggests that teachers tend to focus their attention on specific students or groups, which can lead to unequal opportunities for learning and achievement (Flook, 2016). Additionally, teachers may struggle to monitor all their students effectively, particularly those who are disengaged or disruptive, leading to potential learning gaps and behavior problems (Skinner et al., 2021). Therefore, teachers must develop strategies to manage their attention effectively and ensure that all students receive equal attention and support.

The Classroom Simulator for Teacher-Student Interactions project is a collaborative effort to better prepare teachers for the classroom environment for preschoolers and kindergarteners. Brought together by the vigors of the WPI program, a team of computer science students worked for a collective ten months on developing this simulator for our advisor, Professor Jacob Whitehill. Our sponsor met with the team of students every week to provide feedback to guide the development of the simulator. The team was able to conceive a professional application using the latest technology, one which will be used for years to come for the betterment of our children's future. Ultimately, our simulator will be used to analyze the relationships and interactions between the teacher and student, enabling the user to learn crucial techniques and gain real world experience to the pathway of becoming a fully qualified teacher.

These interactions were programmed to be organic in nature, to capture a 'real life' event occurring on the screen in a random fashion each time the simulation is loaded. The simulator

supports various actions, such as a child running around, the teacher reading to the class, everyone taking a break for snacks, etc. Including a variety of emotions the teachers and students can express, the simulation is continuous and showcases aspects of a typical day in a kindergarten classroom. This is all done automatically by the computer, and cannot be controlled by the user. In short, an extensive amount of information can be learned through this interactive project, and how a teacher should balance giving their attention to not only a handful of students, but the entire classroom.

In this report, the team will describe the initial idea and background of the classroom simulator, an in-depth specification of its key features, the software choices made, and showcase the final implementation of the project. Each section detailed in this report was vital to the team's success and showcases not only our successes of the classroom simulator, but also the greatest challenges the team had to overcome.

The roles of the team were divided up to bring out the best to all of the team's individual strengths. These included but were not limited to software development, learning fluid animation using a game engine, and the brainstorming of the project structure. The team came together to bring this project to life, and without our own previous experience and help from our advisor, the project could have looked very different. Each week a member of the team would be the leader, assigning roles to the other members, discussing future goals and deadlines, as well as leading the meeting with our sponsor.



*Figure #1: First Iteration of Classroom, October 2022*

The first iteration in the image above was the team's effort to create a visual representation of the simulator. Having the avatars' emotions above their heads to display the childrens' feelings as they move around the tilemap was one of the main ideas from the very beginning. The team's next steps were to figure out the best way to animate the sprites and eventually be able to communicate to the user the direction the sprites are facing. It was the team's worry that if the sprites did not turn in the right direction during a said event, it would look awkward and take the user out of the experience.

In the following chapter, the team will dive into the background of this project, and explore other previous simulations our original simulator was inspired by.

## Chapter 2: Background

To give more background on this project, the idea originated from our MQP advisor, Professor Jacob Whitehill, in an effort to help teachers be more equipped to handle certain situations that occur in a children's (preschool through 5th grade) classroom. This involved making the simulation unlike those found in traditional video games and apps today, but one that could be used as a learning tool. This tool, our simulator, meant to be continuous and be different each time it is loaded, provided great insight on the day to day life one could expect of a teacher. Arguably, one of the toughest jobs in the world, a teacher observing a simulated classroom and how the teacher reacts to a variety of escalating situations can better prepare themselves for the real world was crucial to implement in the design.

While researching the best way to implement the idea, it was necessary to seek out other instances of simulators already developed to see the advantages and disadvantages it could provide. Simulators and two-dimensional video games were observed as the team came up with the best way to build our project. Some of the design examples of other simulators, such as the University of Virginia classroom simulator, and two-dimensional games like *Pokemon* and *Legend of Zelda* were particularly interesting. The Virginia simulation (Breen 2016) was three dimensional and had easy to interpret designs for the characters and emotions. It was apparent from the simulator one could tell the emotions of the children, and see their fluid movement as well as their attention span. Other similar simulators had different goals such as platforming, emphasis on dialogue options, and a focus on movement from input by the user. This process took several weeks and finally settling on one software was crucial in the development phase.



Figure #2: Example of Tilemap from Zelda's 2D Display,

<https://www.gamedesigngazette.com/2019/09/how-market-conditions-influenced-2d-zelda.html?m=0>



Figure #3: University of Virginia's 3-D Simulator Capturing Student Response

<https://news.virginia.edu/content/unruly-digital-kids-test-classroom-management-skills-teachers-training>



The team also considered several different designs before ultimately picking the final iteration. It was important for us to focus on the best display that would capture the teacher's attention in regards to the students. Looking at a variety of simulations, the team found several designs that were quite too complex for the goal set out for us. The team also had to consider feasibility and the timeframe of the project, which was an important factor in the final decision. Regardless, the elements that were present in most of the simulations the team observed were implemented in our project. These included structure for pathfinding, tilemap generation, and the ability to restart the simulation from scratch. The team also found many of the simulations were continuous, and the movement was preloaded before it initially started.

During the first weeks of the MQP, our advisor provided the team with a book to give a better idea of the classroom as a whole. An important book that laid the foundation for this project was the *Classroom Assessment Scoring System Manual Notes (Pianta, 2008)*. Provided by our advisor at the beginning of this project, the team all individually read through and saw the different styles of teaching based on a scoring system. During a fifteen minute classroom session, teachers were evaluated on the overall climate based on student interactions. Through this information, the team realized certain areas of learning were graded higher based on the attention the teacher gave to each student. The main goal of the book focused on training teachers to be more precise, and better prepare their observation skills.

In the next chapter, the team will explore the specifications of the simulator and go in-depth on the finer tune details of the project.

## Chapter 3: Specifications

The specifications of the project were built to be reliable for a web browser and easily be usable for preschool to kindergarten teachers. To help them focus more of their attention on certain areas of the classroom, our two-dimensional simulator was built with the goal to make a randomized simulation of events and see how the teacher as well as students react in a standard classroom. On top of this, it will be used to develop various simulations to analyze the attention span and to train teachers to handle these situations appropriately. The job of teaching children, especially children at a very young age, is a big responsibility, and ensuring teachers are aware and trained is fundamental to an adolescent's development. For the project, teachers are able to play through a variety of storylines, watch the behaviors of students as they express their emotions to certain events in the classroom, and collectively observe how the teacher responds. The project was made with the purpose of being simplistic in design, but overall be visually captivating and meaningful for our users. Throughout this chapter, the team will explore the ideas for the specifications used, the possible frameworks the team discussed, and ultimately the technology the team used for the final product.

In our simulator, it is important to showcase a variety of events that can take place in the classroom. To do this, our team created many scenarios that reflect a typical classroom environment and showcase various movements of two dimensional figures with an emotion or action attached. Each scenario plays out as the students can be seen reacting to something or someone, usually involving the teacher. Examples implemented in our simulation included students running around, students making a mess, and the teacher reading a book.

The team aimed to design the simulator to best capture similarities to a typical preschool classroom with layouts that were visually pleasing to interact with. It was important nonetheless

our classrooms looked professional, while also not taking away from the core simulation feature of movement and algorithmically simulated events. The objective for these events was to be determined randomly, so each simulation when generated is unique with their own sequence. Initially, the design process consisted of many user interface mockups, which helped us as a team to further envision what the simulator should look like. In a preschool classroom, there are specific elements almost always present and were made sure to be included in the final product. These elements include but are not limited to at least one carpet, a bookshelf, and tables or desks for students to sit at. Although the layout was lackluster during the beginning phase of the project, the specifications for this design are fundamental in capturing the key interactions between the teacher and their respective students.

The functionality for reading in JSON objects and having it generated within the game engine was a whole phase of the project itself. The team made great progress towards generating storylines, improving movement of sprites, and even made each simulated story unique. By designing the algorithm to take in any randomly generated event, the resulting actions that came from these random choices became its own distinct classroom simulation. The team saved a lot of time by avoiding to hard code each individual story, and in doing so gave many options for the user of this software to use at their discretion. The team ultimately settled for the software to retrieve a randomly generated script associated to a seed number from an Express server, and have the Godot Engine intake and render the information from the TypeScript and JSON object file. The specifications the team had finalized for the software met the qualifications for it to be feasible and simplistic to understand amongst anyone.

In order to create procedural generated classrooms, it was decided to take a model view approach. This approach involved separating our code into two separate portions: the model

portion and the view portion. The model is responsible for generating any rules and parameters that need to be known before a classroom is rendered in the simulator. Some examples include the layout of a classroom, general path vectors of a given student from event to event, and the sequence of events that occur in an instance of a simulation.

Here is an example of what our model will generate for a 10 by 10 room:

Render of Room:

```

w d w w w w w w w
w x x x x x x x w
w x c x x x x x w
w x x x x x x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x x x x x w
w x x x x x x x w
w w w w w w w w w

```

*Figure #4: Output of Rendered Classroom*

The output of a room is currently represented as a 2D array of characters as can be seen above. These characters each represent a different type of tile that is rendered in *Godot* a.k.a our view. In the view portion of the program, the team has implemented a file parser that can interpret this 2D array of characters. The view has a key that will convert each character to a given tile. For instance for 'W' it would be a wall/room border, for 'D' it would be a door, a 'T' for a table, a 'C' for a chair, and 'X' as an empty background tile/floor tile.

Using this key, the view, or rendering engine, can use the index of a given character in the 2D array and match it with the 2D array used for rendering the tile map. The engine or view can

then reference the key and set the given tile in the tile map to the corresponding tile texture from the provided sprite sheets.

### **3.1.1 Student Path-Finding:**

One key aspect of our simulator is the movement of students and teachers between events. In order to accomplish this, the team has developed a pathing algorithm as a part of our model to determine a student's path. The current iteration of this algorithm on the model side works by taking a set of starting coordinates and end coordinates as inputs. Using this it will determine a possible path that can be taken to get from point A to point B without standing on tiles like walls and or tables in an area that can't be occupied by a person in the real world. Here is a small example of a student, highlighted and represented by the char s, moving from coordinates (3, 3) to (3,1) on this 10 X 10 model of a sample classroom.

*Example of a student moving from point (3,3) to (3,1):*

```
Render of Room, Frame: 1
w d w w w w w w w w
w x x x x x x x x w
w x c x x x x x x w
w x x s x x x x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x x x x x x w
w x x x x x x x x w
w w w w w w w w w w
```

Figure #5: Frame 1

```
Render of Room, Frame: 2
w d w w w w w w w w
w x x x x x x x x w
w x c s x x x x x w
w x x x x x x x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x x x x x x w
w x x x x x x x x w
w w w w w w w w w w
```

Figure #6: Frame 2

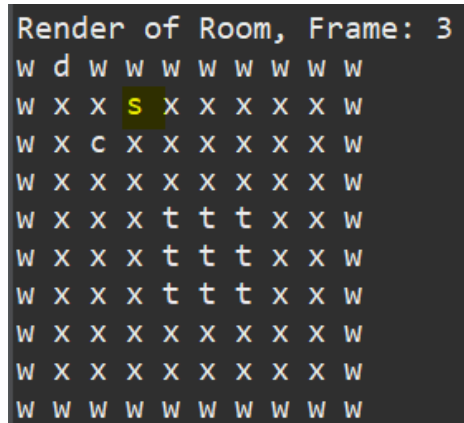


Figure #7: Frame 3

### **3.1.2 Frame Data:**

To build the animation for the simulator, it was important for each frame to implement the specific concepts of Frame Data. In traditional video games or movies there will be a set or targeted amount of FPS or Frames Per Second when it is playing out in real time. A Frame is a single image a combination of multiple frames rendered per second gives the illusion of a moving image or video. For our use case a Frame as referred to in our model is simply a snapshot of the state of a classroom. Combining multiple frames will allow us to visually represent students moving from point a to point b smoothly. Now in the example given above there are only three frames. Three frames per second is extremely slow and does not look very good when played in real time. To solve this problem these three frames will be referred to as data frames. Data Frames directly update the position of an object in the model. Instead of only rendering these 3 data frames and only having an FPS of three, the team has to interpolate the remaining frames between each Data Frame.

For example, let's say there's a one second long event and there are three data frames directly updating the position of something in our classroom. To make the event smooth it is important to use a standard frame rate, typically 60 FPS. Since it's going to render 60 frames

over the span of 1 second the next step is to divide our FPS by the number of Data Frames.  $60$  *Frames Per Second / 3 Data Frames = 20 Frames*. So using this math a data frame for this example will be updated every twenty frames. So for the first twenty frames our student will move from (3,3) to (3,2).

In the next chapter, the team gives an overview of the important decisions made regarding the software and the tools ultimately used for the final implementation.

## Chapter 4: Software Choices

For this chapter, more details are provided on the software tools used to implement the simulator. Initially during the brainstorming period of this project the team strived to look at this simulator from the aspect of a software product. It was important to not only our advisor, but our team that the software could be easily accessible and even accessed as a web application. This meant our code needed to be compiled and compatible in Javascript, HTML, and CSS. At first, this was a minor setback because most of our team had very little experience or were uncomfortable using these languages. In order to save time, the team took advantage of existing frameworks, libraries, or engines to speed up the process of coding and learning.

It was decided earlier on to implement an object-oriented software design approach because this is what was most comfortable for most of the team, but also suitable for the goal of the application. Another important aspect was the extent of documentation and ease of use for future implementations and iterations. Most importantly, the implementations needed to be able to be exported to the web, and work seamlessly in a web environment.

There are two main parts to our app's design. First, it was important to create a model of the simulation. The goal of this model is to create an output that represents and stores important data such as, room size and layout, number of students, and a list of events that pertain to a given simulation. This output created by the model can then be passed into and interpreted by the view for rendering. This model would most likely be done in an Object-Oriented programming language such as Java or Javascript. Next, the team would need a way to serve this model to be rendered on screen. For this it was important to have a view program capable of rendering the output from the model. Ultimately the decision was made to use a game engine that has been built in libraries that will aid in tile maps and image sprite placement. It is crucial the rendering



engine is able to process our model and display it in a fluid and believable manner that accurately portrays the events of a classroom.

## **4.1 Model**

### **4.1.1 Java**

During the first phase of the project, Java seemed to be an obvious choice for a language to build our model in. All team members had experience with it and Java being a strongly typed language with great built in libraries for string manipulation made it an ideal choice. The original idea for the Java model was to create a data structure that represented a given frame, room, and simulation.

Some key components that were developed for the Java model also did not make the final build due to a combination of time constraints or better methods being implemented via typescript model. However, it is important to highlight these components as the team put time into developing them, and they helped us find a better path forward for our final deliverable.

#### CSV File Reading and Writing:

On both the Java and Godot side components were written to read, write, and parse csv files. On the Java side a CSV file would be formatted by overlaying the initial 2D character array that represented a room into a CSV file. Then any events that would alter that initial room would be saved as frame updates in a column similar to that of a chess piece moving on a chess board. This would allow for a rough version of a full simulation to be stored in a CSV file that could be completely generated by the Java model without Godot. This rough version then would be loaded into Godot and data between frames in the rough version would be interpolated. The result was Godot would produce a smooth version of the simulation that makes events such as student movement look more fluid and less blocky. This CSV file that produces the rough

version could also be used as a reference point to rewind to any given frame in the simulation, and continue rendering from that point. Unfortunately due to time constraints the implementation of play pause was put at a lower priority for the project, and this functionality was never fully utilized in the final deliverable.

#### **4.1.2 Vanilla Javascript**

Looking at Javascript, it proved to be a challenge for those who had not written in the language. Unlike Java, team members struggled with the lack of type definitions and looser rules regarding type safety, strict object oriented code, and using a text editor instead of an IDE, which isn't as structured. While writing in Javascript would have meant writing in one language which naturally was accepted by browsers, it would have taken too long for team members to get accustomed to writing in this language.

#### **4.1.3 Typescript**

The model was ultimately written in Typescript. Typescript is a “strict” version of Javascript with declarative type definitions and a more rigid set of rules. Typescript is also built off Javascript, and is built to compile to Javascript and be used on web applications. The type definitions allowed us to build an object-oriented model. It also sped up the design and debugging process because of its extra features compared to Javascript.

#### **4.1.4 Node & Express**

Once the team decided to use a non-native engine to render the animated scenes, it became apparent these scenes needed a server to communicate between the model and view. At first, the team thought of ways to edit the compiled Javascript from the rendering engine, but the

compiled code was not readable and there was logic that could not be changed without breaking the application. Instead, it was necessary to implement an API to communicate the scripts generated by the model with our front-end rendering framework.

NodeJS is a Javascript runtime used for many applications including running servers built with Javascript. Features such as the Node Package Manager (NPM) allow us to easily implement javascript libraries. One library from NPM is Express, which is used to quickly build backend servers. Its extensive use and documentation made it easy to quickly implement an Express API to communicate.

Other advantages of Node and Express are Typescript support, which allowed us to integrate our model directly into the Express application.

## **4.2 Rendering Engine**

### **4.2.1 Javascript Libraries**

Animation libraries included PixiJS, MelonJS and GIMP for the design of sprites and other textures in the simulator.

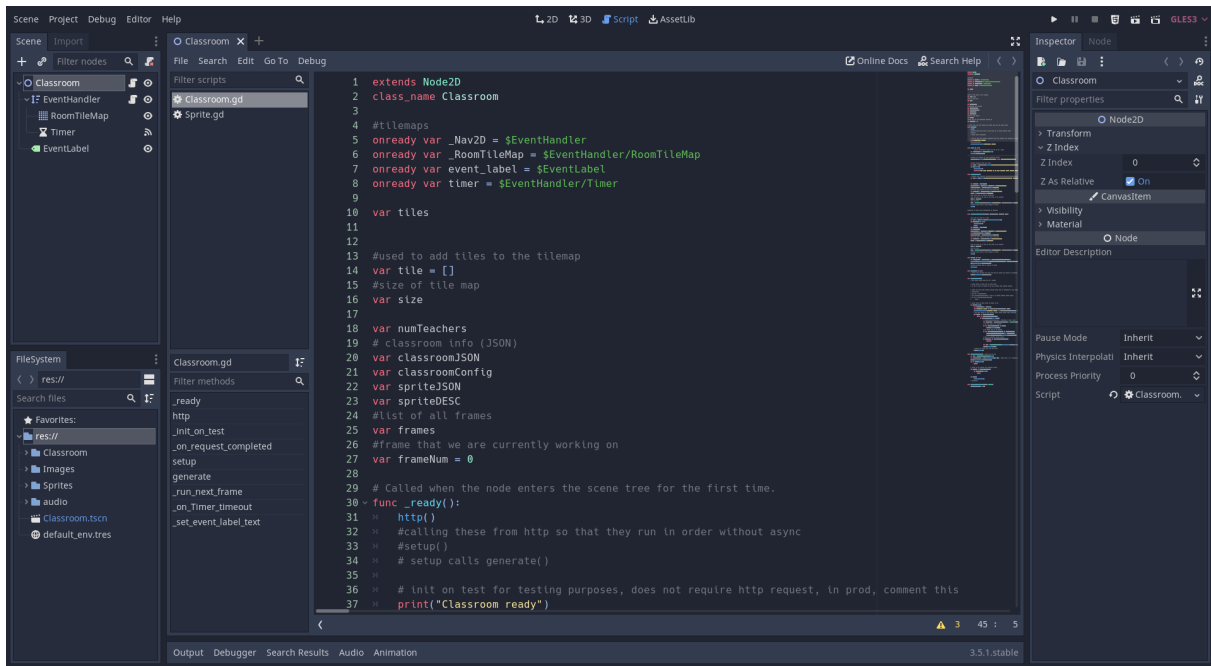
### **4.2.2 Unity**

Unity proved to be too much overhead, had adequate documentation for certain functionality, and was not the best option initially. It was able to compile to HTML though, and proved to be an alternative.

### **4.2.3 Godot**

[Godot's](#) lightweight structure, thorough documentation, and the ability to seamlessly compile to HTML made it top choice for the Classroom Simulator. The usage of Godot had

propelled us to finish our project with all the intended features that the team proposed for in our initial plans.



*Figure #8: This is the Godot IDE which is currently displaying the script view variables and methods are defined.*

Furthermore, Godot's thorough documentation played a vital role in the development of our simulator. The documentation provided clear and concise instructions on how to use the various features of Godot, making it easy for our team to learn and implement the necessary functionalities. In addition, the Godot community provided valuable support through forums and tutorials, allowing us to troubleshoot any issues encountered during the development process.

With its lightweight structure, the team was able to design the layout of our simulator through a framework that is very unique to our project. Godot was the main receptor of information that was being sent from the backend which included information of sprites, tile maps and a script of events that were displayed in the simulator when rendered. Godot's features hosted our team-designed assets that played their role in the rendering.

When using Godot, the team wanted to use the rendering engine in a similar way to making a platform game. The reason for this choice was to follow the guidelines of a platform game for simplicity, due to the intricacy of implementing and designing our own 3D assets. Through this route it was possible to use the dimensions of a 2D grid array where it set locations of sprites and tiles, or randomized every render. Having the option to design our own 8-bit designs helped us with efficiency in terms of time, where the team can focus more on the actual logic of generation with the backend.

The process of having our scripts be generated from server side code to a visual simulation, the team had to associate our assets to a Godot feature known as nodes. With our sprites, it was important to take each one that was designed through Paint Sai or Vanilla Paint and add them to a Node2D (which essentially is a container where other nodes reside that can be 2D transformed via their position, scale, and rotation of their child nodes.)

Within our simulator, sprites are considered as two types, kinematic or static, based on their movement behavior in relation to the physics simulation.

A kinematic sprite is a type of sprite that has a fixed position and can move by changing its velocity. It is not affected by physics forces such as gravity or collisions with other objects, but can collide with other objects and trigger collision events. Kinematic Sprites are often used for player-controlled characters or objects that need to move in a controlled manner.

On the other hand, a Static Sprite is a type of sprite that is fixed in place and does not move or interact with the physics simulation in any way. It is typically used for background elements or scenery that do not need to move or respond to user input or other game events.

By defining sprites as kinematic or static, it is possible to control how they behave within the physics simulation and create a more realistic and interactive game environment.

One of the unique features of Godot is its ability to support multiple programming languages, including node.js and TypeScript. Node.js is a popular server-side JavaScript runtime environment that allows developers to create fast and scalable network applications, while TypeScript is a superset of JavaScript that provides static typing and other advanced features to improve code quality.

In Godot, developers can use node.js and TypeScript code to render out 2D simulations by leveraging the engine's built-in scripting language, GDScript. GDScript is a Python-inspired language that provides an easy-to-learn syntax and powerful features such as garbage collection, built-in signals, and coroutines.

To use Node.js and TypeScript code in Godot, developers need to use the engine's built-in GDNative system, which allows developers to create C++ modules that can be loaded and executed by Godot's scripting language. The GDNative system provides a flexible and efficient way to integrate third-party libraries and existing codebases into Godot games. One of the ways the team used this to our advantage was taking the developed code and being able to export the entire project to HTML5, so it may display through a web-based application view.

Developers can also take advantage of Godot's powerful visual scripting system, which allows them to create game logic and behaviors without writing any code. Visual scripting in Godot uses a node-based interface similar to popular game engines like Unreal Engine and Unity, making it easy for non-programmers to create complex game mechanics and interactions.

### **4.3 Source Control**

Our main way to track all commits from a team of six members was through Github. The team decided this was the best plan of action to keep each other on track. In the following

chapter, the team will explore the details of our main implementation for the simulator, and focus on how it was ultimately manufactured.

## Chapter 5: Implementation

The implementation of the classroom simulator can be split into two parts. The model was written in Typescript and generates scripts which are sent through an express server. These scripts are in JSON format and can be read and rendered in our Godot Engine software. The team used Godot to render the simulation for the reasons explained in the previous chapter. The Godot code compiles to html and can be served to a user through any normal server.

### 5.1 Creating a Story Script

The story script is the output of the classroom simulation and holds all the data and events of the given simulation. Each time the simulator is run, an instance of the Simulator class is created. This class handles generating each event and writing the outcome to the JSON script. It also holds an instance of a Classroom, which is an object that retains the current state of the simulation.

The Classroom consists of a room which is created upon initialization based on the configuration parameters passed through. It also holds a list of Sprites, (students and teachers). Sprites retain information about their current position (relative to the room in x and y coordinates), as well as their current mood and action. At any point, the Simulator can take a “snapshot” of the current state of the Classroom and write it to the JSON file.

A story script includes the room configuration and a list of events that occur in the story. This script is generated by our typescript model based on configuration parameters.

**Table #1: Simulation Configuration Parameters**

| Parameter Name | Description   |
|----------------|---|
| Seed           | A number used in the custom rng elements of the simulation, such as generating events and a room. This allows users to generate the same random simulation using the same seed. If no seed is provided, this will default to a random seed. |



|                |  |
|----------------|--|
| roomSizeX      | Number of tiles in the room that will be in the X direction. Defaults to a random number 9-16. |
| roomSizeY      | Number of tiles in the room that will be in the Y direction. Defaults to a random number 9-16. |
| numStudents    | Number of students in the simulation. Defaults to a random number 4-6.                         |
| numTeachers    | Number of teachers present in the simulation. Defaults to 1.                                   |
| numChairs      | The number of chairs found in the room. Defaults to the number of students + 1.                |
| numTables      | The number of tables found in the room. Defaults to a random number 1-3                        |
| numRugs        | The number of rugs found in the room. Defaults to a random number 1-2.                         |
| numBookshelves | The number of bookshelves found in the room. Defaults to a random number 0-1.                  |

Note that all these parameters are optional and the simulation will use a small range of default parameters in the case that a configuration is not supplied. In practice, it is unlikely that most of these parameters will be used since rooms and scenarios will be generated randomly, but are useful for testing and in the need of a more specific classroom layout.

This script structure was originally designed in Java, but implemented similarly in Typescript when the team changed code bases. Included in the script is a layout of the room, the config, and a list of frames. The state of the room is stored in the form of a 2D character array, as can be seen in *Figure #5*.

```

Render of Room, Frame: 1
w d w w w w w w w
w x x x x x x x w
w x c x x x x x w
w x x s x x x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x t t t x x w
w x x x x x x x w
w x x x x x x x w
w w w w w w w w w

```

*Figure #5: Initial Display of Room using Letters*

This idea was fundamental and would allow for a character to be represented as well as stored in a frame. Each frame would be a snapshot of the classroom simulation at a certain time period. A list of frames would then make up an event, and a list of events would be a simulation. This abstraction would allow for data about a given event, such as a student moving, to be stored as updates to a two dimensional grid. A given simulation could then be logged to a file in an easy to understand format without ever having to open *Godot* or run a simulation all the way through. This meant the team could use this file as a reference point to aid in play, pause, and rewind of a given simulation in *Godot*. Unfortunately, it was decided not to include this feature during the six months of development.

Each frame involves updating the data for sprites (students or teachers) at different points through the simulation. A list of frames is stored by the simulation, creating a timeline of all the events that occur. Upon sending the data from the server, this list of frames is converted into a JSON structure and added as part of the JSON story script.

Events themselves are based on observations of preschool classrooms from public youtube videos. Through taking note of common classroom behaviors the team created a list of

potential actions a student or teacher could take. While this isn't a complete list, the actions created cover many basic scenarios which could occur. Unless a certain event or chain of events is specified, the simulation will generate these events semi-randomly. The team used the term semi-random because some events are more likely to occur if a previous event has just ended. For example, a child who has just fallen asleep will not be able to then run around. A complete list of the events and corresponding rendering behavior can be found in the appendix.

The simulation writes the story script by generating a specified number of events (either randomly or specifically depending on the configuration). After generating each event, The classroom is updated with each Student and Teacher's new data. Then, the "snapshot" is taken of the classroom and converted into a readable JSON object. This object is added to a list of "snapshots" which can be sent as part of the script to be rendered when the simulation finishes. The script is served through an ExpressJS API which was written in typescript. The REST API was created and includes multiple endpoints to suit our needs.

**Table #2: API Endpoints**

| Method | Endpoint  | Description   | Return Type   |
|--------|---|---|---------------|
| GET    | /render-room  | will render a random room with random parameters  | Room          |
| GET    | /render-room/:seed                                    | will render a random room based on the seed you give it   | Room          |
| POST   | /render-room  | will render a semi random room with parameters in config. <sup>[1]</sup> Config is sent in the request body as {config: config} | Room          |
| GET    | /classroom-simulation/random/singleEvent              | runs a single random event with random config, mostly for testing. Right now there are only 2 available events.                 | classroomJSON |
| GET    | /classroom-simulation/random/:numEvents               | will render a number of events with a random config.  | classroomJSON |
| GET    | /classroom-simulation/random/singleEvent/:seed        | will render a single event with a set seed  | classroomJSON |
| GET    | /classroom-simulation/singleEvent/:eventName          | will render a specified single event  | classroomJSON |
| GET    | /classroom-simulation/singleEvent/:eventName/:seed    | will render a specified single event and also a given seed  | classroomJSON |
| GET    | /classroom-simulation/generateEvents/:numEvents/:seed | will render random events given the number of events and a seed.  | classroomJSON |

After much experimentation on the optimal ways to communicate through the API, the team decided GET requests were perfect for our needs in terms of communication. Dynamic routes allow us to send information such as the number of events and a seed to the server by passing parameters through the url. These dynamic parameters are indicated by the “:” in some endpoints. The most common endpoint used in practice was “/classroom-simulation/random/:numEvents”, which generates a specified number of random events. There is no limit to the number of events rendered, and it can be changed simply by changing a parameter. Other endpoints allow us to test specific events or set seeds to replay the same storylines. Using this API allowed us to have maximum control of the simulation while maintaining simplicity and scalability.

## 5.2 Rendering the Script

Using the game engine Godot, the team was able to design and implement a plan to simulate the classroom itself. The reason for this decision was the team's interest in a variety of game engines while also not being afraid to learn something new. The team had previous experience with Javascript, HTML, and CSS, but for the scope of the project the final decision came to Godot, a more reliable alternative. To give a brief background, the game engine provides learnable techniques to use for movement and animations that is easily accessible to the user. The flexibility of the 2D model and resources made available led the team to choosing *Godot*. After overcoming the brief learning curve and brainstorming the best possible way to reach our goal, the team decided on how to best approach the various obstacles associated with the simulator.

Starting from the ground up, the team needed to come up with typical events that would take place in an elementary school classroom. After some consideration, the team came up with over a hundred events involving the teachers and students interacting with one another. To reach the objectives of the simulator, randomness was a key factor in the final product.

Algorithmically, a random storyline is animated and played, with the user having the ability to pause, replay, or even select a specific training. The behind the scenes of this process involved turning this list of events into an animation, which relied heavily on the *Godot* gaming engine. By assigning certain numbers to specific events and effectively randomizing them, the order in which the events play out are different each time. The mindset was to showcase essentially, an unlimited amount of stories each unique in their sequence of actions.

### 5.3 Path Finding

In order to ensure that students did not overlap with solid objects in our Java model the team had to implement a form of two dimensional path finding. It was important to explore a variety of open source third party path finding libraries to aid us in solving this problem. After some time and discussion, the team decided to use the popular A\* algorithm to generate the best path for each specific event. Within the confines of the project, A\* proved to be the fastest and most reliable after a week of testing. Other algorithms such as Dijkstra's and Breadth First Search (BFS) were considered, but the team's past experience with A\* proved to be the optimal implementation moving forward for pathfinding.

Even with the aid of a third party path finding algorithm the team ran into some issues. Certain simulation two students would each be given a path that led down a 1x1 area and get stuck. This would lead to an infinite loop and eventually cause the simulator to crash. The team developed a few solutions to this issue including expanding the edge of the room so that a two unit wide perimeter of open tiles was present in every simulation. Through some testing of an avatar facing a blocked path, the team learned it was possible to generate a new path that it could take after an extensive session of debugging. However, even with these changes there were cases where two students would box another student in permanently, or would get stuck. That on top of the increased complexity being generated by these changes made this solution less than ideal for our project. As a result this path finding algorithm did not make it to the final build, as instead the team used godot built in pathfinding and collision detection to solve this issue without having to alter the parameters of the simulation.

To go further, one of the main challenges for the project was the idea of how to take our Java logic and use it in a web application without creating a complex web server. There were two

paths the group could have taken to solve this issue. The first was to continue using Java and use a local temporary file to store the data on the machine locally. This would have allowed the Java model to write to a CSV file or a text file, and then Godot would read and parse this file to obtain information about a given simulation. The second option was to convert our Java model to typescript and use a middleware such as express to send the data via a web server to godot. The team spent around a month writing in Java, with the idea that the temporary file method would provide a good foundation for the implementation of the potential play pause feature down the line. This is why the decision to switch to Typescript was not made sooner as the team did not have a clear idea of how this feature would be implemented without use of the Java model. The main components and foundations of the model were started in Java, and everyone involved learned a lot during this process. However, the team eventually made the decision to refactor and rewrite our model code in Javascript, as it was easier to convert the model output to godot via a web server than it was to create local temporary files.

## **5.4 Creating the Art**

The next step was designing the classroom and the makeup of the individual avatars as well as their movements. Sprites were made using the *Paint Tool Sai* software.

### **5.4.1 Handling Physics**

Collisions were important to take into account involving the sprites so the story appeared seamless and continuous. If students were walking through tables and chairs, it would distract from the overall experience of the simulator. *Godot* made collisions much easier to deal with, and by programming the sprites to go to a specific location based on a certain event taking place, these problems were avoided. Rendering the sprites and animation was undoubtedly the hardest

part of the project. This step involved several specific scripts that are triggered by certain events in the json file the team created by using the 2D Area feature in *Godot*. This allowed the team to trigger a complete storyline based on the location of the sprites in the simulator and have them move continuously to the next event. Involving emotions and rotations of the figures themselves, the simulator was able to use the gaming engine to fade in, transition, and progress through each event seamlessly. Ultimately, our team figured out how to ensure continuity and create the simulator to play out a five minute story. Consisting of a rather complex algorithm, the details are explored in the implementation chapter.



*Figure #9: Prototype of Simulator (as of December 2022)*

There were many different elements that needed to be taken into consideration when developing this simulator. One of the first major decisions that our team came across was deciding what software to use to create and code the project. Several Javascript libraries were



considered, as well as game engines such as *Unity* and *Godot*. Although there were benefits to each, the team ended up deciding to use *Godot* for our simulator.

Key functionality of the simulator was built by each member of the team using the available tools provided. It was crucial everyone was in constant communication throughout this process, and *GitHub* was used as a central hub for coding documentation. For both *Java* and *Godot*, IDE's such as *IntelliJ IDEA* and *Visual Studio Code* were used to optimize the project. Regular meetings took place to ensure errors were kept to a minimum, and merge conflicts from the integration of code could be resolved. At times, it was difficult to combine all of our unique code into one working model, but the team learned the important lesson of documentation and pushing code in small chunks.

Implementation decisions came down to a group vote throughout each phase of the project. Key factors involved thorough research before the final decision was made. Each decision was equally important as the team received feedback from our advisor to further improve the simulator.

Below is the final implementation of the simulation. The final room featured a doorway, a bookshelf, a couple windows, and a visual representation of a rug. The animation can host six avatars, all with a randomized order of events and emotions.

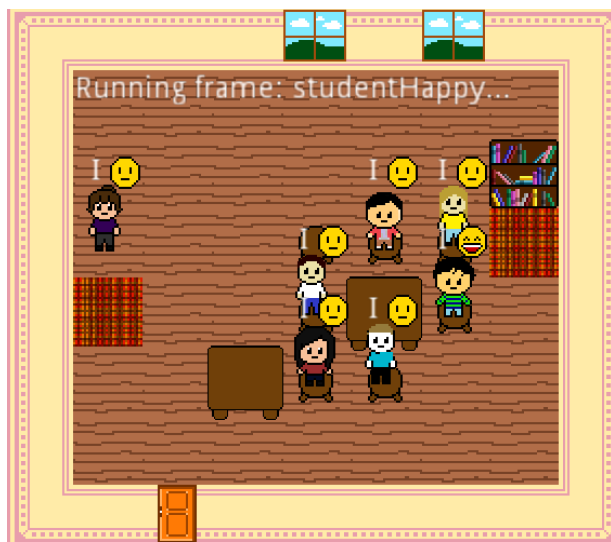


Figure #10: Final Iteration of 2D Simulator

## Chapter 6: Conclusion

The final product for the classroom simulator can be used by Prof. Whitehill's research group for experiments on classroom observation and teacher training. The goal of this project was to provide a simple tool to help teachers learn more about their classroom interactions with students. The next generation of students will need the right care and attention from their respective educators for the betterment of our future. The classroom simulator the team built aimed to take a modest step toward achieving this ambitious goal. The team believes with enough time and attention, anything is achievable.

To best display fluid movement of animated sprites and events occurring pathfinding was key to bring the simulator to life. The integration of *Godot* and code written in the language of Java was difficult but manageable. Features of the pathfinding also had to keep in mind other aspects of the project such as the fast-forward/pause feature. Designing the procedure after the world-renowned algorithm of Breadth First Search(BFS), it made the teacher movement algorithmically process the best path to take for the generated events in the simulator. Collisions were avoidable but by using a set timer, the team was able to capture where the teacher directed her focus for each of the students and events transpiring in an orderly, continuous fashion that made sense.



Creating a visually pleasing tilemap was crucial to create a warm environment for the simulation and user. The color scheme was designed in mind to not be visually distracting but appealing and was modeled similarly to traditional American classrooms. Displaying a random layout of the classroom each time that made sense was complex in nature with the placement of sprites and objects for example like windows or tables. In the beginning, collision conflicts arose as sprites were stuck on certain objects and collided with other sprites. Programming the realism and fluidity for animations was additionally a conflict the team had to overcome but was avoided with the various features *Godot* had to offer.

## **6.1 Lessons Learned**

Going more into detail about some of the lessons learned, for the specifications especially, the team encountered a number of challenges to arrive at our final build. Those included but were not limited to the overall timeline of the project, the learning curve associated with the resources at the team's disposal, and ultimately addressing the client feedback during the workflow to call this project a success. Throughout this project, the team learned the importance of collaboration and working together towards a common goal. Client feedback was equally important and vital to the final specifications for the classroom simulator.

Using the technical components of Java and the powerful game engine of Godot it brought to life the final iteration of the classroom simulator. These tools were crucial in development and testing, while providing meaningful ways to implement our project. If given more time, the team may have gone in different directions in certain aspects of the project. There were many moving components and time of course was a factor in the final product. It was apparent from the first phase of the project, it would be difficult to algorithmically program these moving parts to flow in tandem with the sprites and other objects in the classroom.

To conclude, the team is excited to share the simulator with those who will use it to its fullest potential, the next generation of hardworking teachers. It was our goal the work done over the last year will benefit WPI and the community as a whole. The team learned so much through building this simulator from the ground up, and the knowledge gained from working together will be used in our careers moving forward. The bonds and friendships that came from working towards a working model for this MQP won't be forgotten.

The team is proud of the work they were able to accomplish in the eight months dedicated to this project. Our advisor specifically, Professor Whitehill, was crucial for the success of the project. Coming together as a team to tackle this problem was a great honor, and the team hopes our work is improved on in the future. Thank you to those who made all the resources and software to make this project successful.

## References

1. Breen, Audrey. "Classroom Simulator." University of Virginia School of Education and Human Development. 2016  
<https://news.virginia.edu/content/unruly-digital-kids-test-classroom-management-skills-teachers-training>  
Accessed 21 Mar. 2023.
2. "Java SE Documentation." Oracle, docs.oracle.com/en/java/. Accessed 21 Mar. 2023.
3. "Godot Engine - Documentation." Godot Engine, docs.godotengine.org/en/stable/index.html.
4. Flook, L. (2016). Examining the effects of mindfulness on working memory capacity and attention: A randomized controlled trial. *Journal of School Psychology, 52*, 19-30.
5. Brackeys. "2D Movement in Unity (Tutorial)." YouTube, 19 Feb. 2017,  
[www.youtube.com/watch?v=rycsXRO6rpI](http://www.youtube.com/watch?v=rycsXRO6rpI). Accessed 21 Mar. 2023.
6. Pianta, Robert C. Classroom Assessment Scoring System. Manual. K-3.
7. "Aonuma Says A 2D Zelda Game on the Nintendo Switch Is Definitely a Possibility." My Nintendo News, 14 Feb. 2017,  
[mynintendonews.com/2017/02/14/aonuma-says-a-2d-zelda-game-on-the-nintendo-switch-is-definitely-a-possibility/](http://mynintendonews.com/2017/02/14/aonuma-says-a-2d-zelda-game-on-the-nintendo-switch-is-definitely-a-possibility/). Accessed 21 Mar. 2023.
8. "IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains." JetBrains,  
[www.jetbrains.com/idea/download/](http://www.jetbrains.com/idea/download/). Accessed 21 Mar. 2023.
9. "Download." Godot Engine, godotengine.org/download/windows. Accessed 21 Mar. 2023.
10. Skinner, E. A., Furrer, C. J., Marchand, G. C., & Kindermann, T. A. (2021). Engagement and disaffection in the classroom: Part of a larger motivational dynamic? *Journal of Educational Psychology, 93*(1), 73-85.
11. "PaintTool SAI." Softonic, painttool-sai.en.softonic.com/. Accessed 21 Mar. 2023.

















12. "GIMP - Downloads." GIMP, [www.gimp.org/downloads/](http://www.gimp.org/downloads/). Accessed 21 Mar. 2023.








## Appendix

*Table for Event Description(s):*

| Action:             | Abbreviations:  |
|---------------------|---|
| Breaks toy          | SBT   |
| Draws on object     | SV  |
| Student spills food | SS  |
| Fights              | F   |
| Tumbles             | T   |
| Argues              | A   |
| Bored               |  B     |
| Nothing to do       |  B     |
| Tired               | SLP   |
| Sick                |  SIC |
| Need nurse          | FA  |
| Snack time          | H   |
| Student hungry      | H   |
| Long day            |  B   |
| Happy camper        |      |
| Student happy       |      |
| Happy               |      |
| Student learning    | SL  |
| Focused             | SL  |
| Locked in           | SL  |
| Student sad         |      |



|                                |   |
|--------------------------------|---|
| Student crying                 |    |
| Student upset                  |    |
| Student yells                  | LD  |
| Loud student                   | LD  |
| Student fire drill             | FD  |
| Student needs bathroom         | BR  |
| Student needs a drink of water |    |
| Student wants water            |    |
| Student drinks water           |    |
| Student digests water          |    |
| Student practices writing      |  W  |
| Student writes an essay        |  W   |
| Student studies                |    |
| Student looks at material      |    |
| Teacher timeout                | T/O   |
| Break teacher                  | LUN   |
| Student mad                    |    |
| Student laughing               | xD  |
| Student has a book             |    |
| Student reads                  |    |
| Teacher reads                  |    |

|                                     |   |
|-------------------------------------|---|
| Teacher upset                       |    |
| Student being disruptive            | LD  |
| Teacher losing control of classroom | LCC   |
| Teacher loves job                   | PW  |
| Teacher happy                       |    |
| Teacher teaching                    |    |
| Student raises hand                 |    |
| Student has question                |    |
| Student wants to learn more         |    |
| Student confused                    | ?   |
| Teacher confused                    | ?   |
| Teacher needs more information      | ?   |
| Teacher does not know the answer    | ?   |
| Teacher drinks                      |  |

Event #1: Student Makes a Mess

A student is stationary while being present in the classroom in a random location. The small letter 'SV' appears above the person's head as it moves to the next random event. In the real world for example, a student may spill a drink, or make a mess after eating. This event highlights such activities that typically occur routinely in a classroom, and the events that follow illustrate the consequences of it.



Event #2: Two Students Fight

Two randomly chosen students move while being present in the classroom in a random location. A prompt appears above both involved students with a 'F'. Usually if there is disagreement or bullying taking place, a fight will break out between two students in the real world. Following the fighting, the teacher will be involved to resolve the conflict, and the students will carry on with their scheduled education.



### Event #3: Two Students Hide

In this event, two students move to a random location within the classroom. A prompt appears above both avatars involved with a 'H,' indicating they are hiding. This could be potentially a game of hide and seek or a student practicing an evacuation.



### Event #4: Student Sleepy

A student is stationary while present in the classroom in a random location. A prompt appears above the person's head with 'ZZZ's' indicating a sleepy student. Pictured below one can distinguish a sleepy student from a normal student.



Event #5: Student Sick

A student is stationary while being present in the classroom in a random location. A prompt appears above the person's head with a green emoji. This event indicates that the student is not feeling their best, and may need a trip to a nurse's office.



Event #6: Student Bored

A student is stationary while being present in the classroom in a random location. A prompt appears above the person's head with a neutral face and 'B,' indicating they are bored. In a typical classroom, it is normal for students to daydream and lose interest in the taught material, and this event focuses on that aspect of teaching.



Event #7: Teacher Gets Tired

After several hours of dealing with children with unlimited energy, it is understandable a teacher will become fatigued. In this event, the teacher is stationary while being present in the classroom in a random location. A prompt appears above the teacher's head with a 'SP', indicating the teacher has become exhausted.



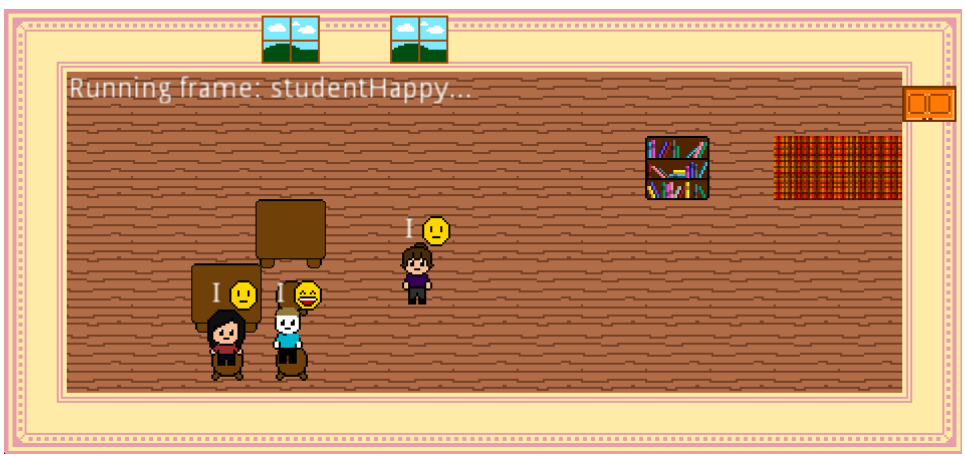
Event #8: Student Eats a Snack

Ultimately, snacks are crucial in learning, and are a welcome break to the important lessons taught throughout the school day. For the eighth event, the student is stationary while being present in the classroom in a random location. A prompt appears above the person's head indicating the student is eating. This event is usually triggered after the teacher becomes tired.



Event #9: Student Happy

For this event, the student is stationary while being present in the classroom in a random location. A smiling emoji appears above the student's head during this event. Whether it be a classmate making a joke, or a student enjoying themselves after a snack, this event is for the user to see the student is currently happy.



Event #10: Student Focused

For the tenth event, the letters 'SL' and a neutral face are the traits used to display when a student is focused. This event is for potentially a student getting ready for an upcoming exam, or an important lesson being conducted by the teacher. Regardless, the user should be aware the student is not easily distracted and has the teacher's undivided attention.



Event #11: Student Cries

A crying emoji appears above the students head for when a student is sad and not feeling themselves. This event is triggered following other events that could potentially lead a student to cry during school.



Event #12: Student Yells

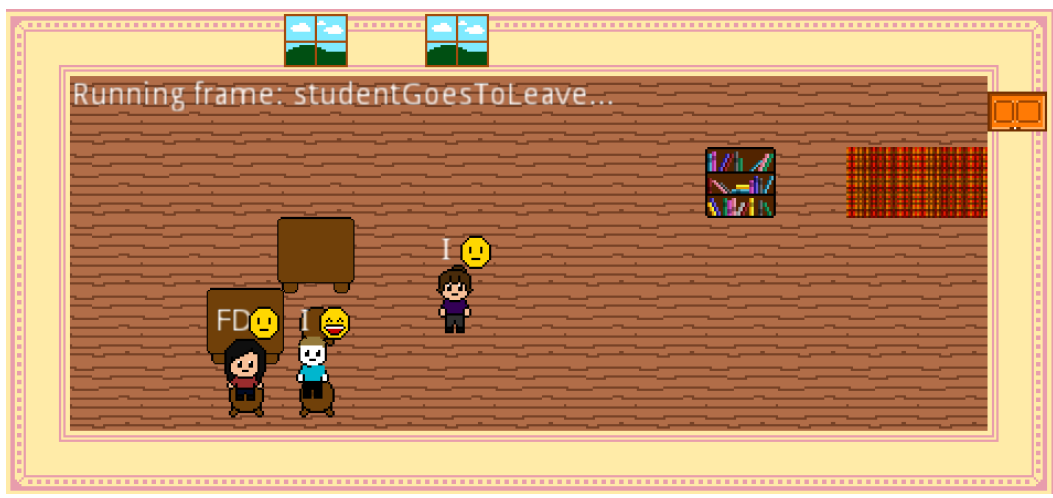
For this event, the letters 'LD ' appear above the student's head indicating they are screaming and/or yelling. Usually in the younger grade levels, it is common a student may become upset or erratic if they don't get their way. This event highlights how the classroom can become out of control if the teacher does not respond to the loud student.





Event #13: Student Goes to Leave

In the thirteenth event in this list, the letter 'FD ' appears above the person's head indicating they are ready to leave the classroom as they move towards the exit. For this event, the students involved don't move initially, but the teacher sees the student may need to potentially use the bathroom, or their parent is ready to pick them up.



Event #14: Student Reads

In a typical school day it is common for the class to practice reading and writing. For this event, a book emoji appears above the student's head (pictured below), indicating the student is reading a book.



Event #15: Teacher Reads

In this event, a book emoji appears above the teacher's head as they read to the class. It is standard throughout the kindergarten curriculum that a teacher reads a story to the class as a whole after a long day of math and science.



Event #16: Teacher Happy

For this event, a smiling emoji appears above the teacher's head, indicating they are happy. Usually following a good lesson, or by the students being focused throughout the school day, this event is for the teachers who find their profession rewarding and take pride in their work.



Event #17: Student Raises Hand

In a typical classroom, from kindergarten to college, students always have questions for the teacher. For this event, a hand emoji appears above the student's head during this event for such a situation.



Event #18: Student Laughs

For this event, a laughing emoji appears above the student's head and can be triggered after most previous events. It is not far-fetched that during the school day for students to keep things fun, a classmate will make a joke or a fool out of themselves. Hence prompting a reaction such as laughing seen by the 'xD' letters.



Event #19: Student Confused

After a long day of school work it is common a student may not absorb all the material taught initially. For this event, a question mark ‘?’ appears above the student’s head, indicating they are confused and potentially may have a question for the teacher.



Event #20: Teacher Confused

For this event, a question mark ‘?’ appears above the teacher’s head indicating they are confused. This event triggers when a student asks a question they may not know the answer to, or by the teacher reacting to the student’s erratic behavior.



Event #21: Teacher Drinks Water

It is very common for a teacher after a long day of talking and teaching for them to need a drink of water. For this event, a water bottle emoji appears above the teacher's head indicating they are keeping hydrated during the school day.



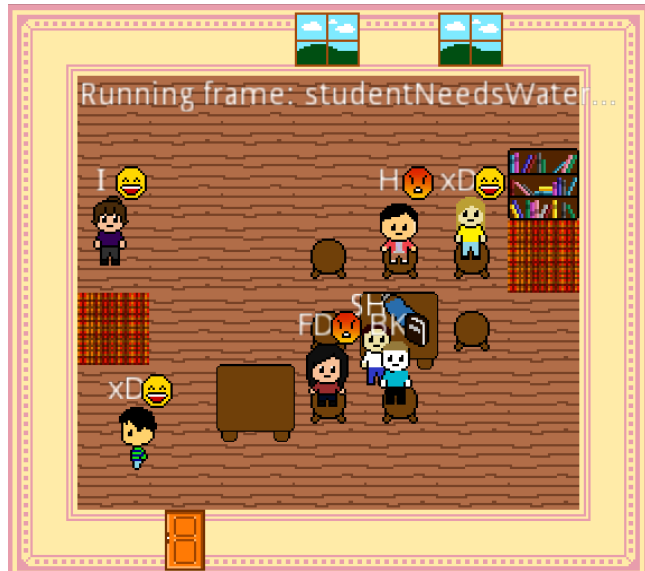
Event #22: Student Needs Water

For this event, a water bottle emoji appears above the person's head, indicating they need water. This event usually leads to the twenty third event, *Student Drinks Water*.



Event #23: Student Drinks Water

For this event, a water bottle emoji appears above the student's head as they drink water for a few seconds. Very similar to the previous event, both indicate the importance of students remaining hydrated throughout the day to avoid dehydration.



Event #24: Student Writes

During a typical school day, a student will practice writing to improve their penmanship. For this event, the letter 'W', and sometimes a pencil emoji, appears above the student's head, indicating they are writing something down or practicing cursive.



Event #25: Student Studies

For this event, the student is stationary while being present in the classroom in a random location. A book emoji appears above the person's head, indicating they are hard at work and studying for a potential exam.



Event #26: Student Timeout

The student moves to a new location in the classroom after a previous event. The letter 'T' that appears above the student's head indicates that they are in timeout.



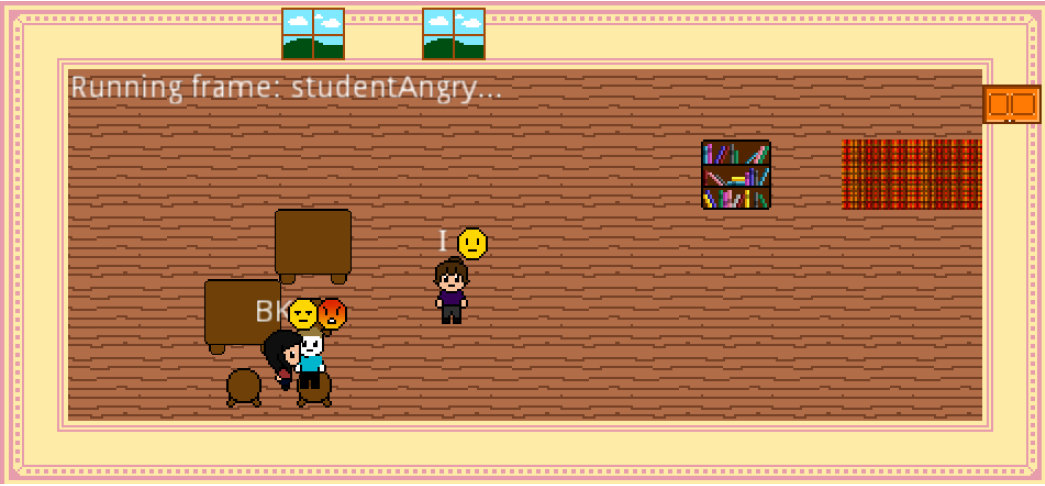
Event #27: Teacher Timeout

The teacher moves to a new location in the classroom in this event. A prompt appears above the teacher's head as they need a break. Displayed with the letters 'TO,' the teacher moves and becomes still for a few seconds before returning to the students.



Event #28: Student Angry

The student is stationary while being present in the classroom in a random location. A prompt appears above the person's head with an angry emoji. This event usually follows a previous event that could potentially anger a student.





Event #29: Two Students Laugh

This specific event involves two students laughing with each other and enjoying the moment. Both students have a smiling emoji displayed above their heads, indicating that they're laughing at something said or the teacher.



Event #30: Teacher Moves to Student

The teacher is initially stationary, and then moves to a new location near a student. A prompt appears above both the teacher and student as they talk in this event.



Event #31: Idle

All characters involved are stationary. Although the events are random each time the simulation is run, the team decided that the sprites start out in an idle state so there is a starting point. Pictured below is the default layout of the classroom with a standard number of avatars present.

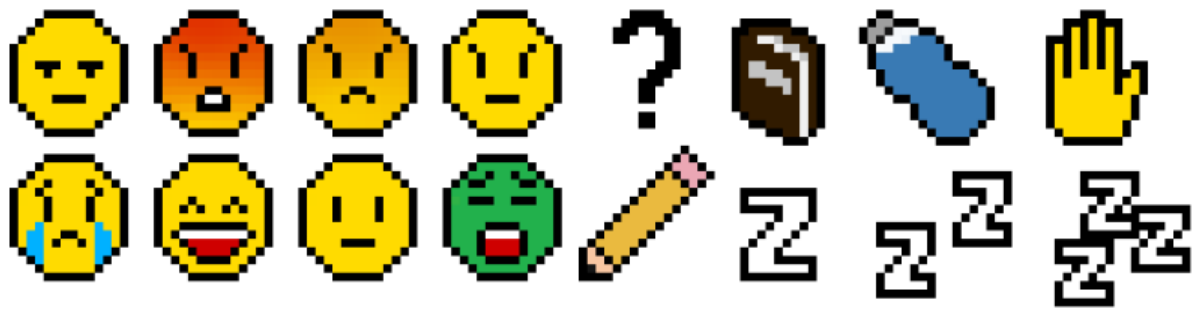
Event #32: Teacher Angry

For this event, an angry emoji appears above the teacher's head, indicating they are displeased with the current climate of the classroom. Usually if a student is chatty or a teacher loses control of the classroom, an angry teacher can be showcased in this animation.

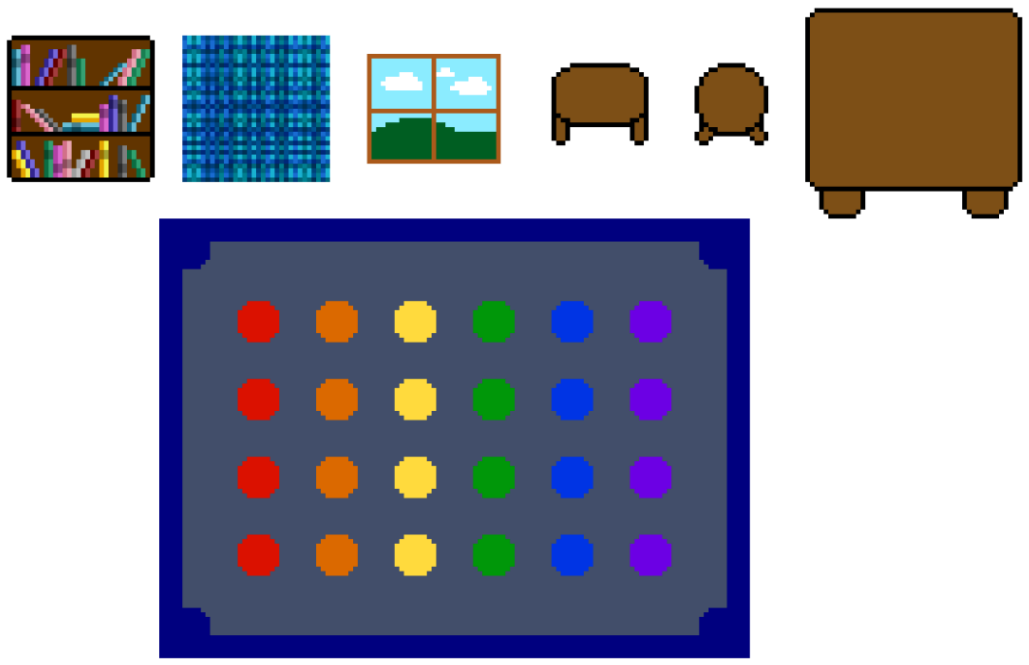


Pixel art:

Statuses: the emotions and actions a sprite is feeling or doing at a given time.



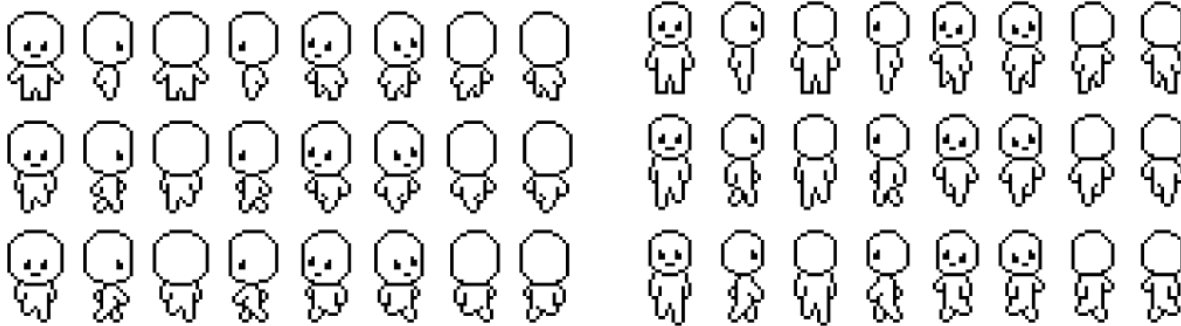
Objects: Every object that can appear in the classroom (top row from left to right: bookshelf, rug – changed to red in final product, window, table, chair, big table; bottom row: big carpet)



Wall designs: designs for the walls and its corners



Sprite templates: templates that each member used to create sprites



Sprites: every sprite that can appear in the classroom

