DESIGNING A VISUAL PROGRAMMING LANGUAGE FOR CREATING MULTIPLAYER EMBODIED GAMES

BY

Matthew Micciolo

THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Interactive Media and Game Development

in the Graduate College of Worcester Polytechnic Institute , 2018

Worcester, Massachusetts

ADVISOR:

Ivon Arroyo

READERS:

Brian Moriarty and Erin Ottmar

ABSTRACT

Games as a means of education have been starting to become more of an everyday reality. Not only are games used in classrooms, but they are used in industry to train soldiers, medical staff, and even surgeons. This thesis focuses on physically active (i.e. embodied) multiplayer games as a means of education; not only by having students play, but also by having students create games. The embodied multiplayer aspect allows for a more interactive experience between players and their environment, making the game immersive and collaborative. In order to create these games, students must exercise their computational thinking abilities. The Wearable Learning Cloud Platform has been developed to enable students to design, create, and play multiplayer games for STEM. This platform allows users to create, edit, and manage the behavior of mobile technologies that act as support to players of these games, specified as finite-state machines. The platform provides a means of testing created games, as well as executing(running) these games wirelessly by serving them to smartphones (or smart watches) so that students can play them with other students, in teams, or as individual players. The platform features a full drag and drop game editor with a sophisticated validation engine that prevents users from making syntax errors. Visually programmed games transpile to JavaScript for execution on the game server and provide two separate levels of programming abstraction. This platform has been successfully tested with 18 participants and they have shown significant improvements in their understanding of Finite State Machines and have shown an increase in their Computational Thinking abilities.

ACKNOWLEDGMENTS

I would like to formally thank the following individuals and organizations who helped make this thesis possible and successful.

PROJECT ADVISOR

Ivon Arroyo

PROJECT READERS

Brian Moriarty

Erin Ottmar

EMBODIED LEARNING GROUP at WPI

MASSACHUSETTS ACADEMY OF MATH & SCIENCE

TABLE OF CONTENTS

INTRODUCTION

In recent years, games as a way of education have been growing in popularity, and a large amount of research has been conducted to show the effectiveness and impact of playing games as a way to learn and practice curricular content (Steinkuehler, Squire & Barab, 2012, Habgood & Ainsworth, 2011, Lester et al., 2013). Most of this research focuses on students playing games as a means of education, while little research seems to have focused on game creation as a means of education. There are benefits to having students “create” games as a way of engaging with STEM content. First, as a way for students to deepen their understanding of curricular concepts --for instance, thinking about math questions from a deep inquiry “problem posing” perspective (Arroyo, Schapira & Woolf, 2001, Silver & Cai, 1996) as opposed to a problem-solving perspective. Second, as a way to have students engage in abstract and systematic thinking, also called computational thinking (Wing, 2006), which implies a high level of precision in the specification of a solution to a problem, which has contemplated all possible movable parts, such as temporal restrictions (what happens first and later), conditional restrictions (what happens in specific situations, as a consequence of a condition), concurrency issues (e.g. states in the game might depend on the states of other players in the game, not only of the previous states of this same player).

The goal of this research project is the generation of a programming language, and a software platform, that would allow students to experiment with both creating and playing multiplayer educational math games that involve the use of mobile technologies to assist players as they practice and learn, while moving around in a highly social and physical spaces. Something unique about this programming language is that it would allow for the “programming” of such mobile devices that support multiple players. Each player carries a device, and the “program” running in the mobile device guides and supports individual players, and/or teams of synchronized players (see Figure 1). By creating these games, participants should engage in a deep process of computational thinking, as well as experience, discuss, and deepen their understanding of mathematics concepts in a community of practice.

In order for students to create these games, a visual programming language and system must be developed and implemented to both give students an opportunity to practice their math skills (as game players) and also their computational thinking skills (as game creators). This programming language has been developed in an iterative manner, after pilot studies that studied both the feasibility of K-12 students both playing and creating these kinds of games, and is the topic of this masters thesis.

This thesis will first describe the context of this research and background research, and will proceed to focus on the architecture of the platform designed to “run games” and the visual programming language designed for children to create games as “finite state machines.” While finite-state automata is considered a fairly abstract concept in Computer Science (Hopcroft & Ullman, 1979), it is actually an intuitive way to represent a system as a series of actions and reactions, and states and transition events between states. This paper shows how we created a simple programming language based on finite state machines for K-12 students to create fun, physically active math games, involving multiple players, math challenges, and math support, building on our knowledge of what a traditional tutoring system would be able to do.

BACKGROUND RESEARCH

Technology In The Classroom

Over the past few decades, technology has exploded into something that we use and interact with every day without even thinking about it. We use it at home, at work, and it is constantly used in schools for educational purposes. When we think of the application of technology in schools, we think of things such as performing research using the internet, writing papers, creating presentations and other stereotypical school tasks, but we forget about many of the intriguing possibilities that could be used in classrooms (Van Horn, 2007). One such possibility is the integration of games into the curriculum and using them as an educational tool. This could include both playing and creating games and could help by adding fun to learning, and in response, make the subject matter much more appealing to students (Micciolo, 2017). By making the games interactive, it could also get students up and moving instead of sitting in front of a whiteboard all day long. According to a 2009 High School Survey of Student Engagement, in which more than 42,000 students participated in, only 2% said they'd never been bored in school and only 41% said they went to school because of what they learn in class (Yazzie-Mintz, 2010). After the analysis of these survey results however, students did indicate what might motivate them. Things such as opportunities to be creative and more engaged were brought up. Games are an excellent way to keep people engaged and by adding the aspect of students creating their own educational games, this allows for them to be as creative as they want, as designing educational games takes a lot of creativity to make a serious topic fun. Games in the classroom could be very beneficial to both the teacher and the students.

Computational Thinking

Playing and creating digital educational games not only has the ability to educate the creator in the topic that the game addresses, but also gets the mind thinking computationally. While playing the games, players may think or be curious about how the game works behind the scenes. More importantly, when creating games, creators must think computationally and programmatically about how they can implement the game they want to. Over the past decade, computational thinking has become a very hot topic and the push to educate students on this topic has been increasing (Shute, Sun & Asbell-Clarke, 2017). Many think that computational thinking only involves programming skills, but it involves more than that such as managing information effectively and efficiently with technologies, so important in our data-driven era. Developing digital games is a great exercise for computational thinking. It can help people get an understanding of programming, algorithms, and event driven tasks. Giving the students an opportunity to experiment with creating these educational games could very well increase their computational skills.

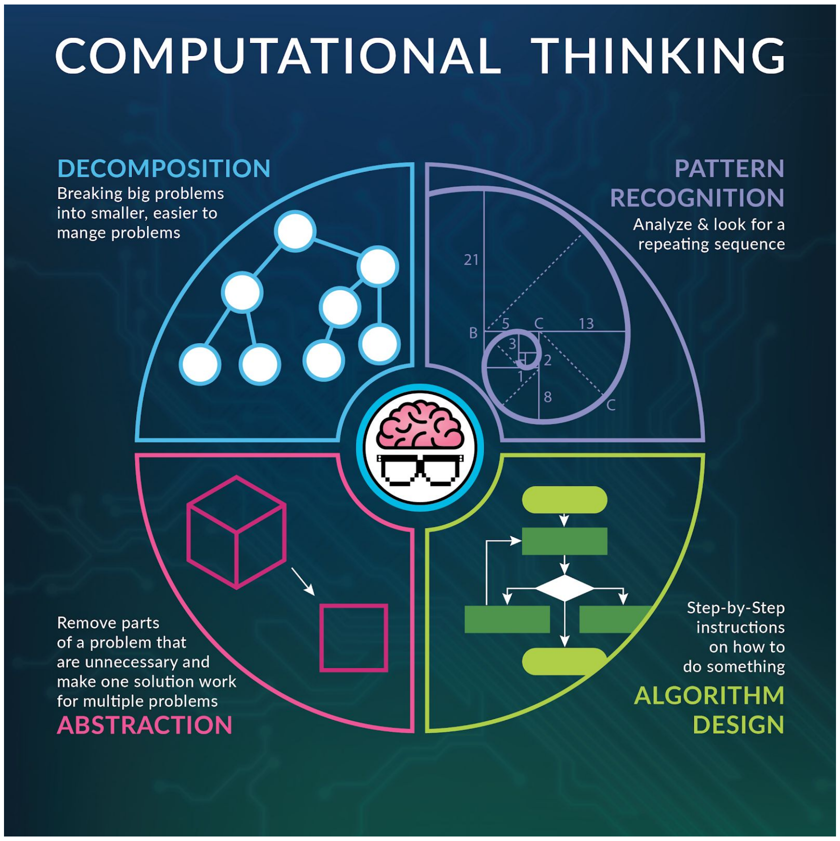
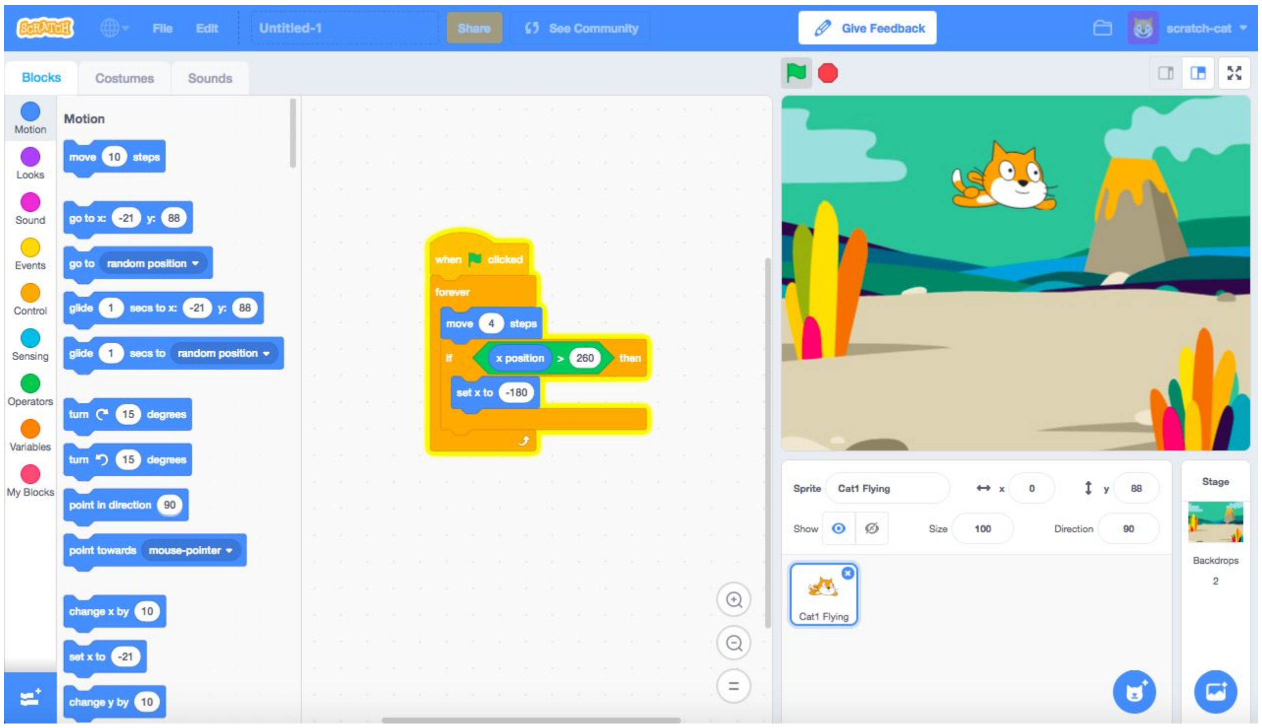

COMPUTATIONAL THINKING
DECOMPOSITION
Breaking big problems into smaller, easier to mange problems
PATTERN RECOGNITION
Analyze & look for a repeating sequence
21
B
5
C
13
3
2
1
8
C
Remove parts of a problem that are unnecessary and make one solution work for multiple problems
ABSTRACTION
Step-by-Step instructions on how to do something
ALGORITHM DESIGN

Figure 1 : Diagram of Computational Thinking (Computational Thinkers, 2018)

Visual Programming Languages as Educational Tools

Over the past few years, there has been a large push to teach computational and programming skills in American schools. Computer science courses for children have proliferated and a 2016 Gallup report found that 40% of American schools now offer coding classes (Tarnoff, 2017). This percent has almost doubled from a few years ago. Teaching kids to program can provide them with skills they need in order to succeed in today's technologically run world. However, teaching them traditional languages such as C/C++ and Java can be difficult and very time consuming for young children. Due to this, programming environments intentionally designed to support novices have become increasingly popular (Price, Dong, Lipovac, 2017). These environments allow students to experiment with programming without having to worry about a complex syntax or about making errors, using a visual programming language that involves dragging and placing blocks, designing their programs graphically rather than textually. At the same time, these programming languages tend to give the user feedback and suggestions. One of the most famous visual programming languages is *Scratch,* created by the MIT Media Lab (Maloney et al., 2010), where children program by dragging blocks that indicate a flow of behaviors, loops, conditional statements, and other programming language functions. The key to a programming environment such as this is its user-friendly nature, so that programming itself can be learned very easily. While one may never use such programming language for a real-world task, it teaches young users how to think computationally, how programs should be organized, and the concept of flow.



```
when [flag] clicked
forever
    move 4 steps
    if x position > 260 then
        set x to -180
```

Figure 2 : SCRATCH Visual Programming Language by MIT (Medium, 2018)

Finite State Machines

A Finite State Machine (FSM) is a method of explaining computer behavior in terms of simple actions and reactions. Finite State Machines can be visually represented by Finite State Machine Diagrams (FSMD). Essentially, a Finite State Machine is an abstract representation of a number of states and transitions between the states that correspond to an input. This behavior can be seen in everyday scenarios such as traffic lights that change when a pedestrian wants to cross the street. In this example, the pedestrian pressing the crosswalk button is an example of input and the change from a green traffic light to a red traffic light is an example of a transition. Synonymous to the traffic light, Finite State Machines can only be in one state at a time, called the current state. The states change when triggered by a condition or event (input), called a transition event. The standard representation of a Finite State Machine Diagram is as follows: states are represented by circles or boxes and events are connected via arrows which represent transitions (Ottmar, Arroyo, Castro, Hulse, Chatani, Harrison & Micciolo, 2017, pp. 3-4). Finite State Machines are part of the Appendix A, the Finite State Machines test given to students to assess their computational thinking ability.

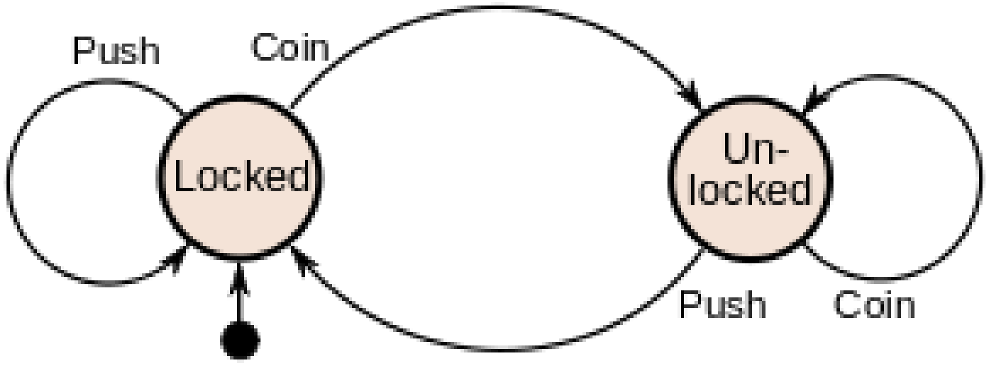


Figure 3 : Finite state machine diagram that represents a locker

Multiplayer Embodied Games for Mathematics Learning

One of the major goals of the research presented here was to create an infrastructure that would allow both the play and the creation of technology-based games that are: a) physically active games so that students are highly engaged and move in a physical space; b) highly social multiplayer games, so that a full class of children can play (think authentic children's playground games such as scavenger hunts, tag, or capture the flag); c) each player carries a mobile device that guides and supports the player, sets challenges for individuals and team for an enhanced 'augmented reality' experience; d) games could be "educational" in that games blend mathematics concepts in them, so that children would have to traverse spaces and encounter challenges; e) players can "input" objects or places found via button pushes, QR scans, RFID scans, camera inputs, or GPS locations.

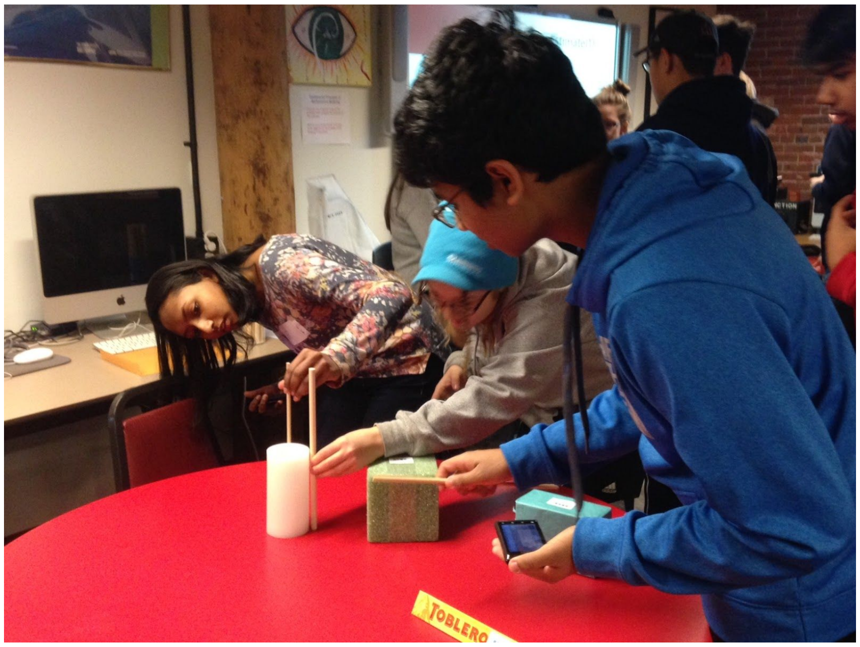


Figure 4 : Students playing the Embodied Math Game EstimateIt!

Game Playing Studies

One of the large feasibility questions that we had, in relation to this new technology and games idea, was whether elementary school students would be able to understand these games in order to play them, or whether the complexity was too much for them. That is, given that each student is equipped with a mobile device that “runs” a game such as a math scavenger hunt, would they be able to understand the goals of the game, understand the role of the cell phone as a scaffold provider and as a guide? Would they understand the role of individual players within a team? Even further, would students learn mathematics from such an augmented reality learning experience?

The result of one controlled study carried out with 53 fourth graders in two schools carried out by a collaborator in the Philippines was an overwhelming “yes”. Participants were 53 students from two math classes in two schools in the Philippines. Students from each school were randomly assigned to one of three conditions: a lecture-only condition, a lecture-plus-game condition, and a game-only condition. The lecture-only group was given a lecture. The lecture-plus-game group was given the same lecture and then played a scavenger hunt game called Estimate It!, where students had to find, measure, and estimate the measurements of target geometric objects in the classroom that had been color coded (and had to be “input” in the cell phones via colored button presses). If students got “stuck”, they could push a button to get help and support. Scaffolds helped them understand the concept underlying the task, and got them closer to the solution (target object).

The game-only group played EstimateIT! but did not experience the lecture. Results suggested that the groups playing EstimateIT! obtained the highest means in math post-test performance scores, and that the lecture-only group obtained the low-est means in math post-test performance. This showed the feasibility of such technology-based games to be implemented in schools, and that at this game was an activity conducive to student math learning, boosting the learning potential of a lecture covering the same material. Also, students really liked the active embodied math game. When asked if the game was fun, 93% of students said the game was “fun”, citing reasons such as physicality (e.g. "the game not only makes me learn, it also makes me exercise"), and social interaction (e.g. "it's fun because I'm playing with my friends"). 86% of students said they would prefer playing the EstimateIT! game over usual classroom instruction. Some students asked whether the game was available for download in the app/play store. The results in general suggest that student experience was extremely positive (Arroyo et al, 2017; Casano et al., 2016).

Other studies in the United States and China also showed that such active learning game experiences, with students carrying individual devices that would give them support, are conducive to learning. For instance, students playing the Tangrams Race, a relay race that encourages teams of students to take turns to find target geometric objects in a basket at an end line, and then run back to the starting line, followed by construction of a puzzle with the retrieved pieces, led to significant performance gains in standardized test math questions, after only a single session of playing the game (Arroyo et al., 2016).

The Wearable Learning Cloud Platform is infrastructure that allows the creation and playing of such distributed games, and facilitates research on active embodied student learning in such collaborative and competitive experiences.

Past Game Creation Studies

We wanted the Wearable Learning Cloud Platform to enable the easy creation of new games; a game creation environment that is simple enough for teachers, and even for middle school children to create these active games. One of the questions we had was whether middle/high school students would be able to manage to conceive and create such physically active math games. On the one hand, younger folks who have played playground games only a few years earlier should probably be able to imagine and conceive fun multiplayer games, even better ones than adults would. However, would they be able to manage the level of abstraction? Would they generate interesting and feasible math game experiences that go beyond the EstimateIT! scavenger hunt we had created? Would they manage to represent them and specify them with a high level of precision?

In order to answer this question, fifty-four (54) sixteen-year-old students from a high school in Massachusetts were involved in a “math game design” study. Students worked on a variety of activities during three one-hour sessions, three consecutive days in 2017. Students worked in teams to create, draw, and specify multiplayer active math games on large paper pads, given the following constraints: a) The game you design must be playable by 4-6th graders (9-12 year olds); b) The game has to teach students some math concepts appropriate for their age; c) The game must be multiplayer (at least 6 simultaneous players); d) They should make sure the game is active by getting the students to move around, at least to some extent.

In addition to this, students played EstimateIT!, with the purpose of having students understand the role that mobile devices could play as guides and assistants/tutors in the math game. Students were explained how the game they just played worked as well the system behind it powering it. Day three was completely different than day one and day two in that the entire group redesigned the games they had created before to include mobile devices, similar to EstimateIT! (guiding and supporting the players). Last, students were given a presentation that summarized the idea of finite state machines (finite state automata), and how actually the EstimateIT! game was internally specified as a finite state machine, from start to finish, with states representing outputs and button presses as inputs and transitions between states.

Students proceeded to redesign their games to include cell phones, and specified the behavior of these as finite state machines. After they were done redesigning their games, they presented their new games to each other. After each day, students were given a homework assignment, writing a two-page summary of what they had produced each day. Other submissions to this conference show that students were able to conceive these games, include technology, and specify them to a large level of detail as finite state machines, with a good level of precision. Thus, the result to the feasibility of at least high school students creating these games was extremely positive (Arroyo et al., 2017, Hulse et al., submitted; Harrison et al., submitted). Students were able to create a large variety of games that went beyond our initial scavenger hunt, assign a role for the cell phones, and specify the phone behavior in detail using finite state machine diagrams.

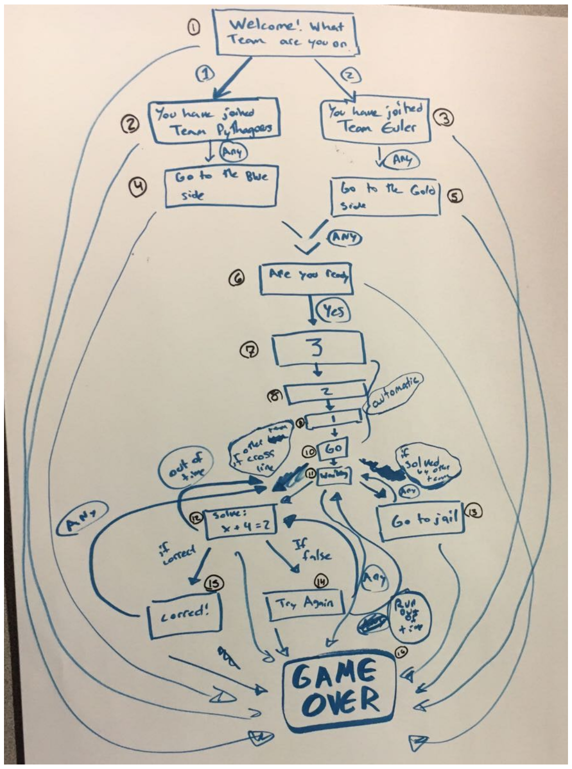


Figure 5 : Student created paper FSMD

RESEARCH QUESTIONS

As said before, the goal of this research project is the generation of a programming language and a software platform, which I named as the *Wearable Learning Cloud Platform*. This platform allows students to experiment with both creating and playing multiplayer educational math games that involve the use of mobile technologies to assist players as they practice and learn, while moving around in a highly social and physical spaces. By creating these games, participants will engage in a deep process of computational thinking, as well as experience, discuss, and deepen their understanding of mathematics concepts in a community of practice. The following research questions will try to be answered with this master's thesis project.

- Regarding Students' Computational Thinking:
    - Does a person's knowledge of defining a computational system with Finite State Machines (one way to assess students' Computational Thinking ability) change by designing multiplayer math games, both on paper and in the WLCP?
- Regarding the visual Programming Language:
    - Is the WLCP suitable for use by K-12 students?
    - Is it possible to create a user friendly and easily usable of designing and playing finite state machine games digitally?
- Regarding Students' Coding and Game Development Processes:
    - How well can someone translate a game they designed on paper into a Finite State Machine Diagram on paper?

- When going from paper FSMD games to WLCP FSMD games, did users encounter limitations?

METHODS

This Thesis is comprised of 3 parts: The Wearable Learning Cloud Platform, a study and an analysis of that study. The Wearable Learning Cloud Platform part is comprised of many small submodules that provide a means for users to design, create, and play multiplayer games for STEM. It is comprised of a Visual Programming Language Specification, Data Model, Game Editor, Game Server, Transpiler and Web App. The next part, the study, will consist of students volunteering their time to test designing, creating and playing games using the game editor and visual programming language. Lastly, results from this study will be analyzed and a conclusion made about how effective the system is.

Developing a Visual Programming Language

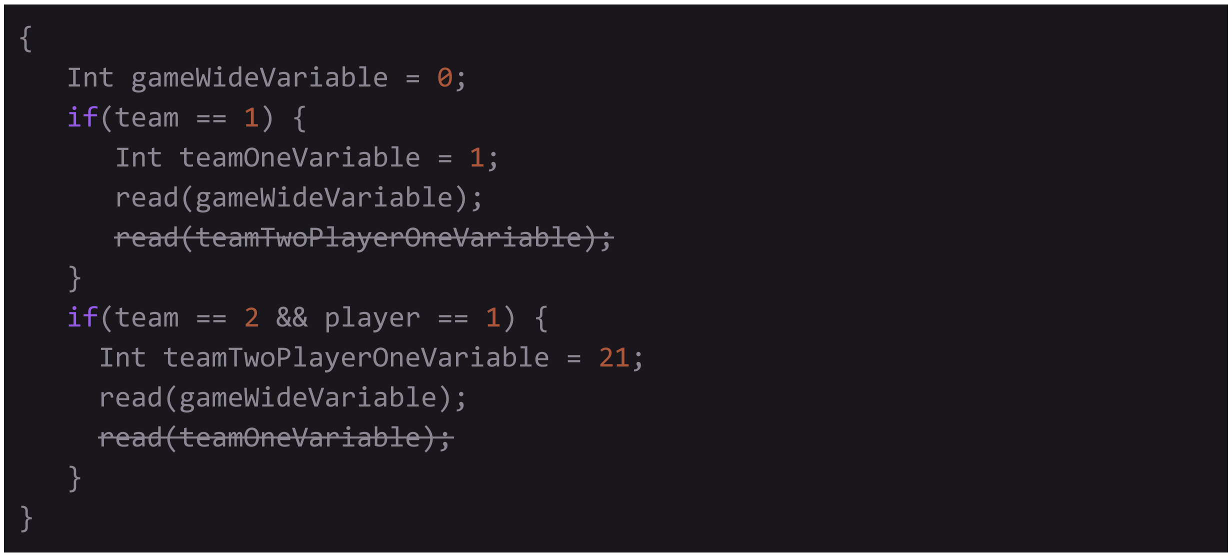
WLCP is a visual programming language whose underlying model of computation is a Finite-state machine (FSM). The programming language consists of 3 main components: states, connections and transitions. States exists as a mean of output such as displaying text on a screen. Connections provide a path to another state to transition to. Lastly, transitions provide a way to control connections via inputs such as button presses, keyboard input, rfid scans, etc.

Multiplayer Aspect

What makes this visual programming language more unique than others is its ability to define real time multiplayer games. This has to do with how the data is defined and treated when using the programming language. The language contains three scopes at which data can be accessed : The Game Wide Level (Global), the Team Wide Level and the Player Wide Level. If data is defined at the Game Wide Level, then all clients running program have access to that data. If data is defined at the Team Wide Level, then all clients of the same team have access to that data. If data is defined at the Player Wide Level it is only accessible by an individual client (player). The following works in reverse; Player Wide Level can access Team Wide Level and Game Wide Level and Team Wide Level can access Game Wide Level.

Multiplayer Scope

The multiplayer scope is similar to how scope works in a traditional language such as C/C++ and Java. Below is an example of how scope works in a traditional textular language. This is C/Java pseudo code.



```
{
   Int gameWideVariable = 0;
   if(team == 1) {
      Int teamOneVariable = 1;
      read(gameWideVariable);
      ~~read(teamTwoPlayerOneVariable);~~
   }
   if(team == 2 && player == 1) {
     Int teamTwoPlayerOneVariable = 21;
     read(gameWideVariable);
     ~~read(teamOneVariable);~~
   }
}
```

Figure 6 : Traditional textual language scoping

On the first line a game wide variable is declared. Then in each of the if blocks (which represent team scope and player scope), is the declaration of two more variables, one for the team if and one for the player if. Each of the if statements can read the variable defined at the top, just like how teams and players can both see the game wide scope. However, the team 1 if cannot read the team 2 player 1 if statements declared variable because it is out of scope. The same applies for the opposite. The same applies for this language. When data is defined at the team level, you cannot define it at the player wide level.

States

Because the WLCP Language is based on the concept of Finite-State-Machines, states need to be defined; thus, states are represented as boxes with a label that describes itself. States also contain one input and one output node (represented as circles in the top center of the state and bottom center of state respectively). The input node may contain single or multiple input connections and the output node may contain single or multiple output connections.

States contain variable and output data. The data is defined for the three different scopes as described in the Multiplayer Aspect section. Whenever a state is transitioned to, its variables and output data is enabled for the scope of the current client (player). These different types of output data include but are not limited to text, images, sounds, light, etc.

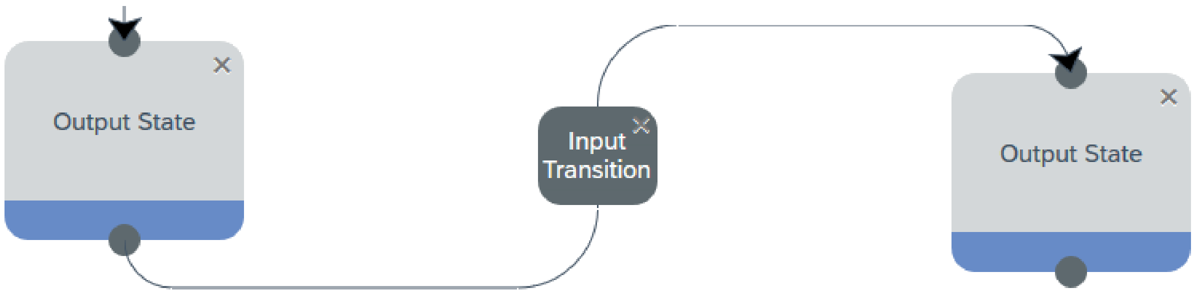


Figure 7 : WLCP Visual Programming Language. Consisting of States, Connection and Transitions

Connections

In the WLCP Language, connections are represented as lines with one arrow that connect from one output node to one input node. A single connection may not contain more than one transition, but does not have to contain any.

Self Loopback Connection

It is possible to make a connection from a states output to that same states input, thus creating a self loopback connection. If the connection does not have a transition attached this creates an infinite loop. If there is a transition attached this creates a controlled infinite loop.

Loopback Connection

A connection whose output does not equal the same states input is defined as a loopback connection. It behaves the same as the Self Loopback Connection, but does not loop back to itself, just from another state.

Neighbor Loopback Connection

A connection whose target is a neighbors (share a same parent state) input.

Transitions

In the WLCP Language, transitions are represented as small boxes that reside on the center of a connection. A transition can only ever belong to one connection. Transitions work exactly in the same way as a state with the exception that they work as inputs and thus contain input data rather than output data. The different types of input data include but are not limited to single button presses, sequence button presses, keyboard input, RFID Scans, GPS location, Accelerometer, etc.

Wearable Learning Cloud Platform

The main goal of this section is to describe the Wearable Learning Cloud Platform (WLCP), a system designed to support the creation and deployment of multiplayer embodied games via a web-based platform. The WLCP provides the following services: (1) communication with the devices; (2) maintains the state of individual players and games; (3) aids the users (game manager) in the general functioning of activities (start the game, verify progress, determine the winner); (4) management of game lobbies of players; (5) keeping track of individual and team progress; (6) creation of new games through a “game editor” based on finite state machine diagrams. Any user can login to any of the three main modules of the WLCP, which are game manager, game editor, and game player.

Architecture

Before actual development started a decision had to be made on how the project should be setup and which tools should be used in order to implement the WLCP. In order for the system be accessible by everyone, regardless of which machine or operating system they were using, a decision was made to make everything web based. The next decision was which tool suite to use to build this web based platform. Java and JavaScript were selected. Java would be used for backend development and JavaScript would be used for frontend development.

For development of the platform, the cloud approach was taken. This means that the software is hosted by one entity and accessible everywhere as long as you have access to the internet. This means that users don't need to install any extra software or host it themselves. They simply access it from their web browser using a PC or mobile devices such as tablets and phones.

The platform can be broken up into several parts which include: Java Persistence API (JPA) Data Model with MySQL, HTML5 web based UI using SAP OPENUI5 (game manager, game editor, and virtual device), multithreaded asynchronous TCP socket Java game server, JavaScript transpiler and SPRING web app. More detail about these various components will be revealed in the following sections. Below you can see a diagram of the WLCP which is comprised of its backend technical components, frontend client presentation components and lastly, the game play component of students building and playing games.

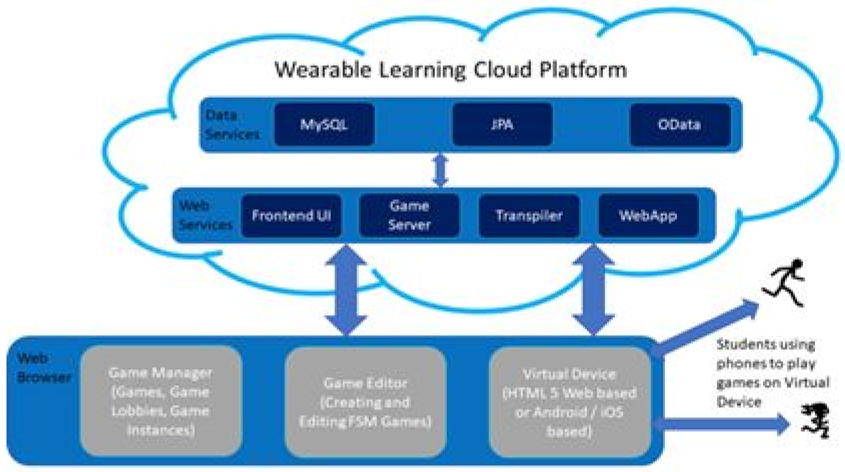

Wearable Learning Cloud Platform
Data Services
MySQL
JPA
OData
Web Services
Frontend UI
Game Server
Transpiler
WebApp
Web Browser
Game Manager (Games, Game Lobbies, Game Instances)
Game Editor (Creating and Editing FSM Games)
Virtual Device (HTML 5 Web based or Android / iOS based)
Students using phones to play games on Virtual Device

Figure 8. Architecture of the Wearable Learning Cloud Platform

Testbed

For development, testing and study purposes a physical server was needed to host the WLCP and all of its components and dependencies. An HP ProLiant DL165 G5 was selected. The server contain dual AMD 4 core processors, 16gb RAM, and a 4 x 73gb RAID ARRAY in RAID 5. The server was setup in the Atwater Kent building with the permission of Worcester Polytechnic Institute and given a static ip.

Host Setup

Ubuntu Server 16.04 was installed as the host operating system. From there, KVM and Libvirt were configured to provide virtualization and virtual Network Address Translation (NAT). The virtualization allows for emulating different computers running different operating systems and the virtual NAT allows for emulating a software router between all of the virtual machines. This allows for many servers to be run on a single piece of hardware and logically split up in the network like they were independent physical machines. Apache Web server and BIND DNS were also configured. Apache was setup to act a reverse proxy and forward web request to other virtual machines.

Virtual Machines

Four virtual machines were then configured. The web VM, wlcp vm, seafile vm and the desktop vm. The first three were configured with Ubuntu Server 16.04 and the last with Ubuntu Desktop 16.04.The VMs were split up based on the services that they had to provide since this testbed has multiple purposes. They are described in detail individually in the following sections.

Web VM

The purpose of the Web VM is to handle web traffic going to embodied.wpi.edu and its other sub cnames. To do this it uses Webmin and Virtualmin. These two packages provide a way to manage the server remotely as well as setup new accounts for hosting various domains.

WLCP VM

The WLCP VM handles hosting an instance of the Wearable Learning Cloud Platform. This includes a MySQL database server, TOMCAT Web Server, Java Game Server, Jenkins Server and Nexus Repository Manager Server.

Seafile VM

The Seafile VM runs an instance of Seafile. Seafile is a cloud file sharing application that enables the storing and retrieving of files via the web.

Desktop VM

The Desktop VM provides an instance of Ubuntu Desktop 16.04. This is used for managing the other virtual machines as well as accessing internal resources behind proxies for maintenance and configuration.

Data Model

The first and most important part of developing the WLCP was to design the data model. The data model is responsible for storing and providing all of the data for the entire platform in an organized and easily modifiable way. This includes all of the different relationships between entities as well. For example, a parent entity may have a relationship to a child entity. Most databases are designed using an SQL language or some other API that allows for the creation and management of the relational data.

Selected Tools

Since most of the application was going to be written using Java or a web scripting language such as JavaScript, the choice was clear to use the Java Persistence API (JPA) instead of an SQL language to design the database. JPA is provided by Java out of the box allows one to design a database by placing annotations on different classes and members of those classes. Using another library called EclipseLink enables the automatic generation of SQL code and thus automatic database generation for all of the major databases such as MySQL, PostgreSQL, etc. MySQL was chosen as the database for this project as it is very widely used and supported. The project would be called WLCPDataModel in the repository.

Design

With the database and persistence API chosen, it was time to start designing some of the tables, the relationship between the tables and the items in the tables. The data model was split up into four main categories : master, state, connection and transition. Inside each of these categories are classes to describe different tables.

Master

Master contains all of the definitions for tables that will store master data. Master data is defined as core data that usually stays pretty static and doesn't get modified much. The WLCP has 4 tables defined in master data which include : game, game instance, game lobby and username.

The class (table) username stores information about an individual user of the system. Things such as username id, password, first name, last name and email address. These are all defined using Java standard types such as String, Int, Float, etc and annotations such as @Id (defines primary key) and @Column (defines this attribute as a column in the table). JPA also allows for fields to be defined using lists, maps, etc. For example, username also contains a field that is a list of game lobbies (will be explained later) the username owns. Lists can be defined using the @JoinTable annotation which creates a separate table to manage the list items.

The next class (table) is game lobby. A game lobby is defined as a group of players that can play a game together. They will be described in more detail in the Game Manager section. This class contains the fields game lobby id, game lobby name, username and a list of usernames that are in the lobby. JPA also allows referencing other tables via their class type.

The game lobby class contains a username that is defined using the java class. JPA maps this to the primary key of that username by using the @JoinColumn annotation.

The next class (table) is game. This contains information about a game created in the editor. This includes the fields: game id, team count, player per team, username, visibility, state count, connection count, transition count, data log, list of states, list of connections, list of transitions. Most of these fields are just references to parts of the game stored in other classes (tables).

The last class (table) is game instance. This contains information about instances of games running on the game server. A game instance can be defined as a game lobby playing a game. This includes the fields game instance id, game lobby, game, username, debug instance.

State

State contains all of the classes (tables) required to store the data for a state that is placed in the game editor. In the WLCP there are two different types of states; the start state and an output state. A start state stores no data and simply signifies where the program begins. An output state stores data about the different types of outputs it has configured. Due to the fact that these two types share similar data such as position in the editor, etc a base class (table) called State was configured using the @Inheritance annotation which joins the base with either the start state class (table) or output state class (table) depending on the type that the state is defined as.

The state class (table) stores all of the base data for every single state. This includes the following fields: state id, game, state type, position x, position y, list of input connections, list of output connections.

The start state class (table) does not contain any additional fields to the state class. It does however set the state type field of the super class to START_STATE.

The output state class (table) contains one additional field which is a string to string map called display text. This field maps the scope of the output state to the display text it has been configured with. In order to to do this an @ElementCollection and @CollectionTable and @MapKeyColumn annotation were used to create a new table which maps an output state to its display text map entries.

Connection

The connection category includes a single class (table) called connection. This purpose of this class is to store all of the different connections between states for all of the different games. It contains the following fields: connection id, game, connection from, connection to, backwards loop and transition. The connection from and connection to fields are references to the state where the connection comes from and the state where the connection goes to. The transition field is defined as null if there is no connection on the transition. If there is, it is set as a reference to that transition in the transition class (table), which is described next.

Transition

Transition contains all of the classes (tables) required to store the data for a transition placed in the game editor. This includes a transition class, single button press, sequence button press and keyboard input. The transition class contains the following fields: transition id, game, connection, active transitions map, single button press map, sequence button press map, keyboard input map. Active transitions are a map of the scope to the transition type that is currently being used (single, sequence, etc). The single button press map maps a scope to another class (table) which contains the single button press data. This is the same for the sequence button press and keyboard input map.

The single button press class (table) contains four booleans for four different buttons that could be pressed for this transition to succeed. The sequence button press class (table) contains a list of sequences that can be input for the transition to succeed. Lastly, the keyboard input class (table) contains a list of input strings that can be input for the transition to succeed.

```
@Entity
@Table(name = "USERNAME")
public class Username implements Serializable {


    private static final long serialVersionUID = 1L;

    @Id
    @Column(length = 40, name = "USERNAME_ID")
    private String usernameId;

    @Column(length = 40, name = "PASSWORD")
    private String password;

    @Column(length = 40, name = "FIRST_NAME")
    private String firstName;

    @Column(lengt = 40, name = "LAST_NAME")
    private String lastName;

    @Column(length = 40, name = "EMAIL_ADDRESS")
    private String emailAddress;

    @JoinTable(name = "USERNAME_GAMELOBBIES", joinColumns = @JoinColumn(name =    "USERNAME_ID",
referencedColumnName = "USERNAME_ID"), inverseJoinColumns =   @JoinColumn(name = "GAME_LOBBY",
referencedColumnName = "GAME_LOBBY_ID"))
    @OneToMany(orphanRemoval = true)
    private List<GameLobby> gameLobbies = new ArrayList<GameLobby>();
}
```

Figure 9 : Username JPA Table Definition

Test Data

The test data project was developed as a tool for use during development and during deployment. This project is a very basic web application whose sole purpose is to generate the databases schema from the JPA data model and then populate that newly generated schema with some test data. Test data can be defined as made up usernames, games, game lobbies, etc that are created for use during development to help speed up the process. This project contains one main servlet called TestDataLoader. When a HTTP request to this servlet is made, a connection is made to the database. Next, eclipselink generates the databases schema and tables from the JPA data model. Lastly, data is loaded from CSV files and placed into the appropriates entities and tables.The database is now generated and populated with data for use during development. This project can also be used to initially generate an empty schema when deploying the WLCP to a new host.

Main Interface

With the data model now designed and developed, the main interface could now begin development. This main interface would be web based and serve as three services: Game Manager, Game Editor and Game Player. The game manager would be used for managing game lobbies and starting and stopping games. The game editor used for creating and editing the games using the WLCP visual programming language. Lastly, the game player would serve as a client for playing the games on.

Selected Tools

Java and JavaScript were the main tools used for the implementation of the WLCP. Java was be used for the backend web app components and JavaScript for the frontend presentation layer. For the frontend presentation layer, SAP OPENUI5 was chosen. This is a fairly new HTML5 JavaScript UI framework with a modern look, modern features and rich data binding characteristics. The framework relies on the Model View Controller (MVC) design pattern. The model is represented using JSON, views are designed using XML files and controllers are programmed using JavaScript. For the backend web app the following frameworks were chosen : Apache OData and SPRING. OData provided a REST service in which the frontend data model could bind to and access data in the database. SPRING provides a similar service and other things and has mostly replaced OData in this project. This project would be called WLCPFrontEnd and WLCPWebApp in the repository.

Design

Index

When a user first accesses the main interface via any web browser, they are presented with an index page. This is default page that everyone navigates to. Once a user has navigated to this page they can then select whether they would like to be logged into the game editor, game manager or game player.

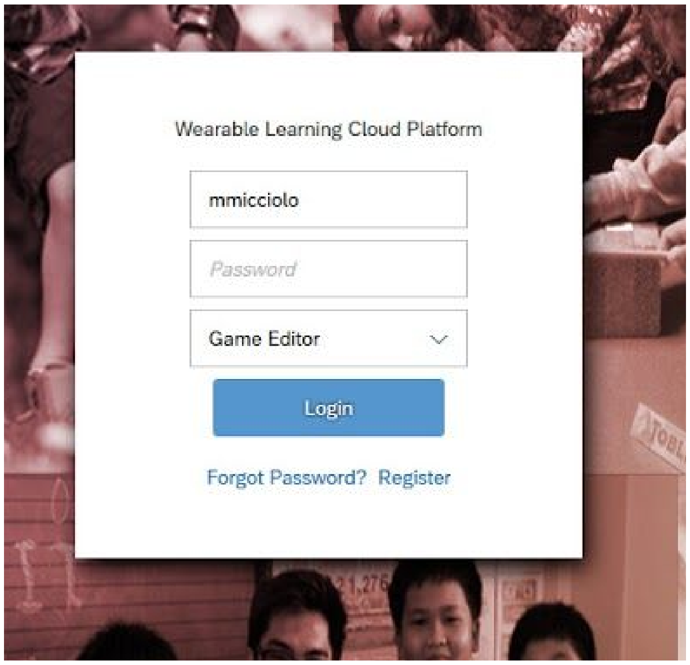


Figure 10 : WLCP Index Login Screen

Game Editor

The game editor's purpose is to provide a means for users to create, edit and debug games using the finite state machine based WLCP visual programming language. The interface relies heavily on dragging, dropping and clicking. The editor also features a validation engine that will automatically enable and disable options based on how the game is configured. This is done in real time and allows for instantaneous feedback to the programmer. Data is provided to the game editor via SPRING controllers in JSON format.

General Layout

The editor consists of a toolbar at the top that allows for creating new games, loading, saving, and running (testing and debugging) them. When the new button is pressed a dialogue is presented and one must fill out the game name, team count, players per team count and if its visible to the public or not. Clicking load opens up a new dialogue with a drop down where one can select the game they want to load. Pressing save opens a waiting dialogue and saves the game in the background. Both the loading and saving use a SPRING controller to load the data as JSON and save the data from JSON. A SPRING controller is used because it can automatically translate JSON to and from a JPA data model, which is what is being used for the data model. Lastly, clicking the run and debug button saves the game, transpiles the game and then starts a debug game instance on the game server and opens a game player window. In a debug instance, any team or player can be selected and you don't have to setup a game lobby and manually start an instance making it quicker and easier to develop and debug games.

It also consists of a toolbox on the left-hand side that allows users to drag states and transitions onto the screen. The remaining space is the editor pad, where the game can be made. States can be moved around by dragging while holding left click. Connections can be made from dragging off an output end point while holding left click and dropping onto another input endpoint. Transitions can be dragged onto an already existing connection. Users can remove states and transitions by clicking the X in the upper right hand corner. A connection can be removed by dragging it off a end point while holding left click.

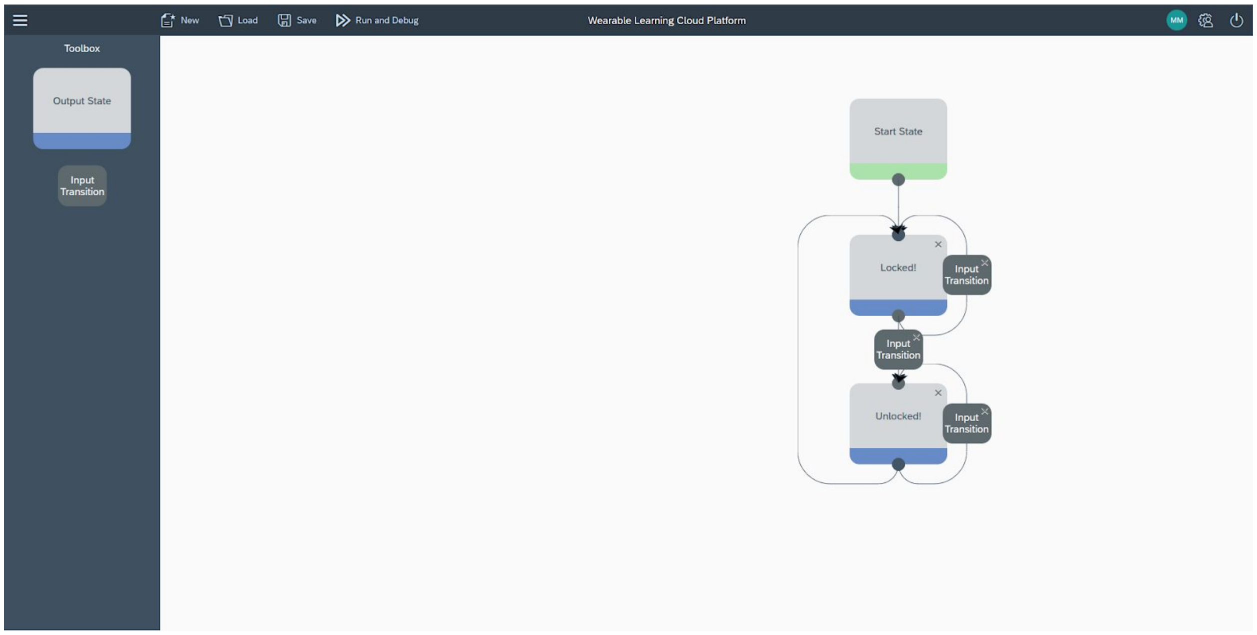


Figure 11 : WLCP Game Editor

State & Transition Config

Double Clicking on an output state (does not work for start state) or input transition causes the configuration dialogue to open. The configuration dialogue for the both states and transitions consists of a scope selection box at the top, a type selection box to the left, a configuration page to the right an accept button and a cancel button. The scope selection box contains an aggregation of icon tabs which allow users to select either the game wide, team wide or player wide scopes. An output state has an additional input field at the top called state description which is used as the value for the text displayed on the output state in the editor canvas.

Once a scope is selected a state or transition type can be selected. An output state can be configured with a display text, which displays text on the game players screen. This can be input into a text box on the configuration page. A transition can be configured with a single button press, sequence button press or keyboard input. A single button press configuration page contains 4 check boxes with either red, green, blue or black. Ticking any of them means if that button is pressed, the transition will go to the next state. A sequence button press configuration page has a list of sequences and an add button. Adding a sequence requires one to drag a red, green, blue or black tile in an order and submit. It can be any length long. When the sequence is put in correctly on the game player, the transitions goes to the next state. Lastly, the keyboard input configuration looks the same as the sequence button press configuration but instead of dragging a sequence in a specific order for the configuration, you type in text into an input. Entering the correct text into the input on game player causes the transition to go to the next state. For both the sequence button press and keyboard input, if an empty sequence is configured this means that the transition will trigger if the sequence or keyboard input from the game player is not contained in the list of sequences or keyboard inputs configured for that scope and transition.

After a state or transition is done being configured, the user has the option to select the accept or cancel button. Clicking the accept button confirms the changes and notifies the validation engine to revalidate the appropriate states and transitions. Clicking the cancel button reverts the state or transition to the configuration values it had when it was first opened.

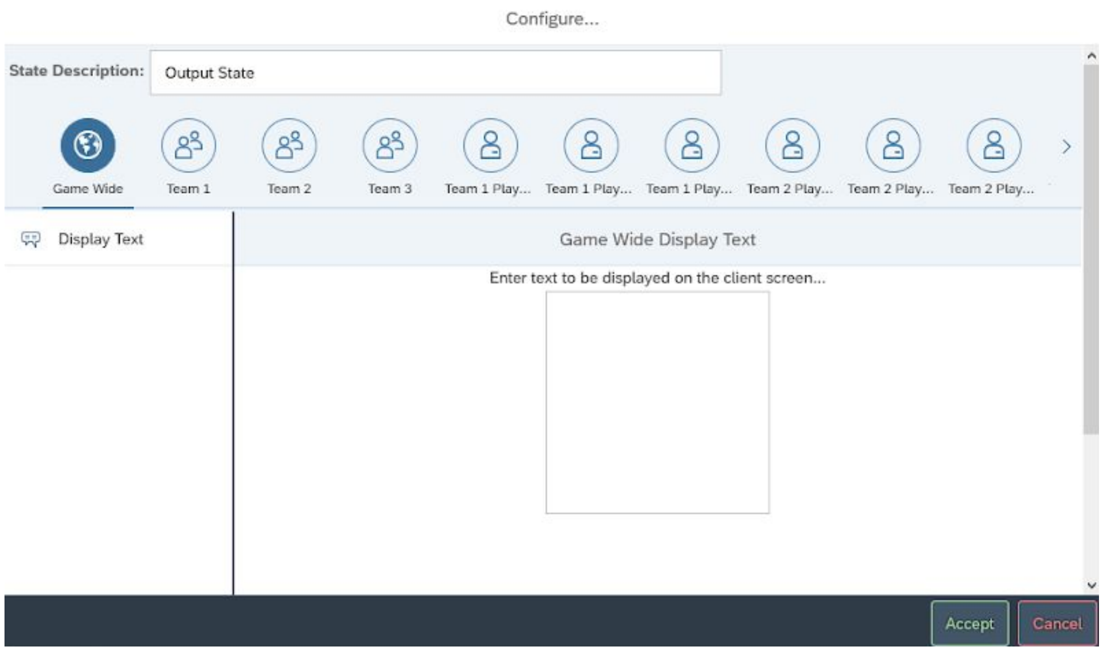


Figure 12 : WLCP State Config. Transition Config is the same but with Single Button Press, Sequence Button Press and Keyboard Input

Validation Engine

Every programming language has to have some sort of feedback for the syntax. For a textual based programming language, the compiler does this by checking the syntax of the code as it compiles and stops if there are any errors. For my visual language, I chose to build this into the code editor rather than into the JavaScript transpiler. This offers a few key advantages for the user such as: they cant make syntax errors and won't have to deal with debugging them and as changes are make to your code, the syntax is revalidated live and you options of what you can choose are modified depending on what you do. This essentially makes it impossible for a programmer to make a syntactical error that would cause the game not to transpiler or play correctly.

In order to do this a validation engine had to be built for the game editor. The engine would be tasked with dynamically revalidating a state or transitions scope based upon what is configured for that state or transition and also based upon what is configured for its parent, neighbor, etc state or transition. For example, if the game wide scope is configured all of the others would disappear. If team 1 was configured, the game wide scope and team 1 player wide scopes would disappear. If team 1 player 1 was configured, the game wide scope and team 1 scope would disappear.

Game Manager

The game manager is responsible for managing the playing of the games that were created in the game editor. It provides the following services : managing game lobbies, games and game instances. These services can be accessed by using menu on the left hand side of the screen. Data is provided to the game manager via the OData service from the web app.

Game Lobbies

A game lobby is defined as a group of usernames that can play a game together. Game lobbies were created as a solution to an existing problem. When a game was started, anyone using the system could see every single game running. This could potentially make it hard to find the game you want to join is many are running and also could allow for unwanted users to join the game. In short they act as a filter so that when a user is trying to join games, they will only be allowed to join a lobbies game if their username is in the lobbies list.

When you click on the game lobbies menu item, you will be redirected to a tile interface showing all of the game lobbies you have created. You can create a new one by clicking on the green plus button. If you click on a game lobby tile, the user editor screen is brought up. From here one can add users and delete users to and from the game lobby respectively. The interface also provides a means to search for game lobbies by name and create them via an uploaded CSV file.

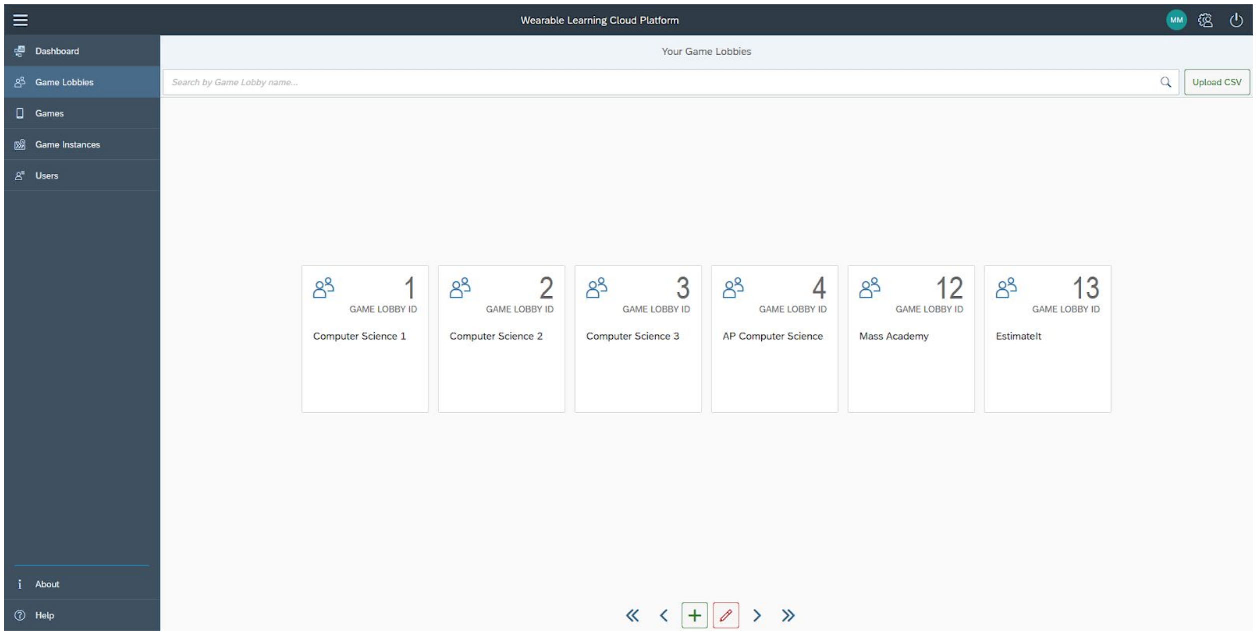


Figure 13 : Game Lobbies Screen. Create, Edit and Delete Game Lobbies.

Managing Games

In the game manager, you can manage games created in the editor. This screen uses the same tile interface that the game lobbies screen uses. The functionality is limited to the ability to viewing games by name, delete games by name and navigating to them. Creation of games is only supported in the game editor. If a user wants to delete a game, they can click the red trash can button and then the X button on the tile that corresponds to the game they want to delete. Clicking on a tile will automatically redirect the user to that game in the game editor. A search bar at the top provides an easy way to find games.

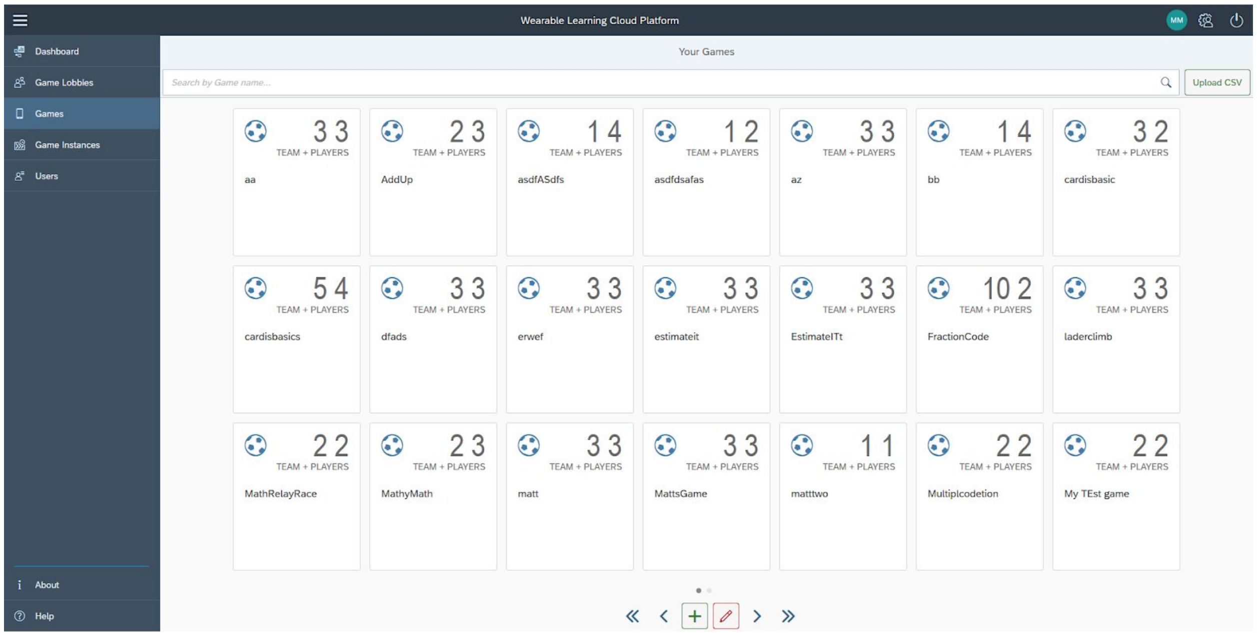

Wearable Learning Cloud Platform
MM
Dashboard
Game Lobbies
Games
Game Instances
Users
About
Help
Your Games
Search by Game name...
Upload CSV
3 3 TEAM + PLAYERS aa
2 3 TEAM + PLAYERS AddUp
1 4 TEAM + PLAYERS asdfASdfs
1 2 TEAM + PLAYERS asdfdsafas
3 3 TEAM + PLAYERS az
1 4 TEAM + PLAYERS bb
3 2 TEAM + PLAYERS cardisbasic
5 4 TEAM + PLAYERS cardisbasics
3 3 TEAM + PLAYERS dfads
3 3 TEAM + PLAYERS erwef
3 3 TEAM + PLAYERS estimateit
3 3 TEAM + PLAYERS EstimateITt
10 2 TEAM + PLAYERS FractionCode
3 3 TEAM + PLAYERS laderclimb
2 2 TEAM + PLAYERS MathRelayRace
2 3 TEAM + PLAYERS MathyMath
3 3 TEAM + PLAYERS matt
3 3 TEAM + PLAYERS MattsGame
1 1 TEAM + PLAYERS matttwo
2 2 TEAM + PLAYERS Multiplcodetion
2 2 TEAM + PLAYERS My TEst game

Figure 14 : WLCP Games. Can edit in editor and delete.

Managing Game Instances

A game instance is defined as a game lobby playing a game on the server. This screen provides a means of starting and stopping game instances. This screen is used when a user wants to start and game with others and connect using the game player to play it. The game instances screen is organized the same way as the game lobby and game screen. Currently running game instances are shown as tiles and a search bar at the top provides an easy way to find game instances. When the user clicks the green plus button and user is presented with a dialogue in which they must select a game lobby and what game for that game lobby to play. When the click start, the game is started on the WLCP game server and can be played using the game player. To stop a game instance one must click the red trash can and then the red X on the tile corresponding to the game instance they want to stop.

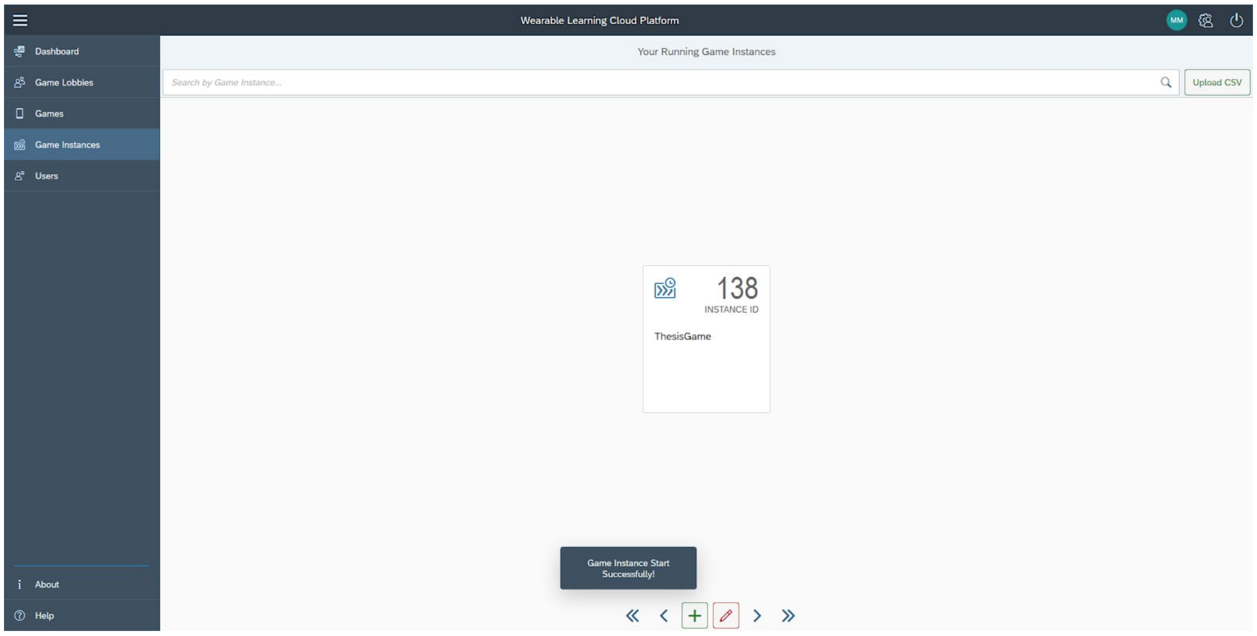

Wearable Learning Cloud Platform
MM
Dashboard
Game Lobbies
Games
Game Instances
Users
About
Help
Your Running Game Instances
Search by Game Instance...
Upload CSV
138
INSTANCE ID
ThesisGame
Game Instance Start Successfully!

Figure 15 : WLCP Game Instances. Can start and stop game instances. Requires a game and game lobby with users.

Game Player (Virtual Device)

The game player (virtual device) acts as a user client that is interacted with when playing a game. All communication between the virtual device and game server is done using web sockets and a custom shared packet system which will be described in more detail in a later section. Using the game player consists of the following steps: selecting a game instance, selecting a team, playing.

After logging in and selecting game player, a drop down appears that contains a list of game lobbies that you belong to that have a game instance running. After the user selects the game lobby they want to play in they then select what team they want to be on. If the team is full it won't show up. The player has now successfully joined a game and is presented with the virtual device.

The virtual device is broken up into three main components : the title bar, outputstate page and the input transition page. The title bar has the text virtual device and a disconnect button. The output state page contains a text box that display any output text. The input transition page changes depending on what type of transition has been configured in the game. It will either be four static buttons for single button press, a drag and drop interface for sequence button press and an input field for keyboard input. This behaviour would be the same for output state if there was more than just a display text output.

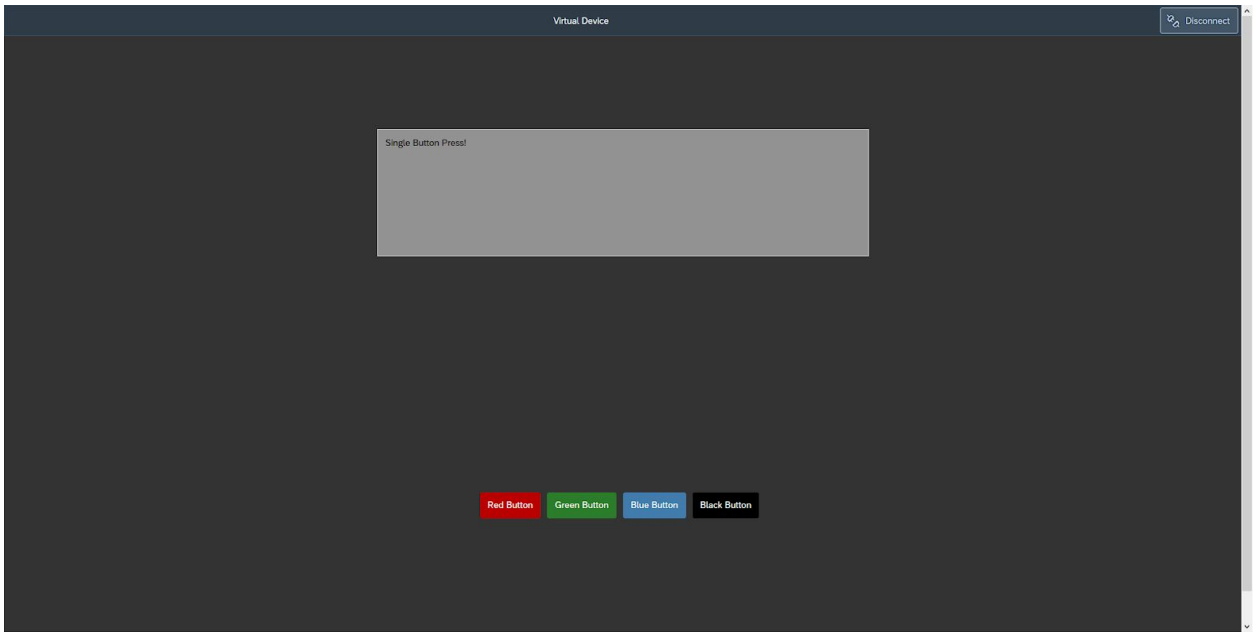


Figure 16 : WLCP Game Player (Virtual Device)

Transpiler

The transpiler is one of the most important parts of the entire Wearable Learning Cloud Platform. It is responsible for the important task of translating the games from their data model representation into another form that can be executed on the game server. In this case, it is transpiled to JavaScript which is then executed on the game server using the Nashorn JavaScript engine.

Transpiling from the data model to JavaScript was chosen over other methods such as writing an interpreter for the data model for several reasons. The first is that by using JavaScript you get a lot of features that would have had to been implemented from scratch, for free. Some of these things include branching, looping, variables and expression evaluation. Also, by designing a transpiler, one could easily transpile these games to other languages and thus run them on other servers designed by others. Another benefit is it allows more advanced users to design games using JavaScript. Lastly, by transpiling to an intermediate language such as JavaScript one can start to understand how a textual languages works just from working with a visual one first and seeing how it transpiles. Since education is one of the main focuses on this project, this provides an additional step of education being able to see how your visual representation could be represented in code.

Selected Tools

Since most of the other applications were being written using Java it was only appropriate to design the transpiler using Java as well. This would allow the transpiler to easily be triggered from a call to a web app controller. It would also allow for some reuse of code that is used in order to access the data model and read its entities. Other than that, all the transpiler does is produce a textual JavaScript file, so not much more was needed.

Design

Transpiler Template

The first thing to design for the transpiler was a code template that could be filled in with different code and data depending on how states, connections and transitions were configured in the game. Since the visual programming language is finite state machine based, it was decided to design the template as a finite state machine but using JavaScript. Below you can see the empty template.

```
var states = {
   thesis_start : 0,
};

var FSMGame = {

   gameInstanceId : 0,
   team : 0,
   player : 0,
   playerVM : null,

   running : true,
   state : states.thesis_start,
   oldState : null,

   start : function() {
        while(this.running) {
        if(this.state != this.oldState) {
        this.oldState = this.state;
        this.stateMachine(this.state);
             }
        }
   },

   stateMachine : function(state) {
        switch(state) {
        case states.thesis_start:
        this.thesis_start();
        break;
        }
   },

   thesis_start : function() {
   },

};

var SetGameVariables = function(gameInstanceId, team, player, playerVM) {
   FSMGame.gameInstanceId = gameInstanceId;
   FSMGame.team = team;
   FSMGame.player = player;
   FSMGame.playerVM = playerVM;
}
```

Figure 17 : The transpiler template

The template above is broken up into six main parts : 1. State enumeration, 2. Namespace and variables, 3. Start function, 4. State machine, 5. State machine functions (states), 6. Set game variables. The purpose of the state enumeration is to map the state id to an integer, so it can be converted from one to the other. The namespace is basically just a global variable that contains the entire program. Inside of the namespace are some variable declarations such as the gameInstanceId, current team and player the player is, current state, etc. The start function is the main loop of the game. It continually checks if the state variable has changed (due to a transition) and if so, calls that next state in the stateMachine. The state machine function simply calls the correct state function based off of the value of the state variable. Lastly, the set game variables method provides a way for variables in the JavaScript to be set from Java code.

It is important to note that the transpiler generates a state function such as the one below for every state that exists in the visual representation.

```
   thesis_state_2 : function() {
   }
```

Figure 18 : A state function that the transpiler generates for every state in the FSMD

These functions are filled with calls to the PlayerVM (described later) to trigger different outputs and inputs. In the snippet below we can see the state requesting the VM shows some text on the devices screen.

```
   thesis_state_2 : function() {
        this.playerVM.DisplayText("Hello World!");
   }
```

Figure 19 : A state function that displays text

In this snippet below, we see a SingleButtonPress call being made to the VM to request input from the device, in this case a single button press of red, green, blue or black. The code below specifies that either 1 or 2 will transition to state 3. This corresponds to either a single press or red or green.

```
   thesis_state_2 : function() {
        this.playerVM.DisplayText("Hello World!");
        this.state = this.playerVM.SingleButtonPress(["1", "2"], [states.thesis_state_3,
states.thesis_state_3]);
   }
```

Figure 20 : A state function that displays text and encodes a button push transition event

In this snippet, we see a SequenceButtonPress call being made to the VM. This works similarly to the single button press, except the press value is a sequence of buttons. For example, 12 represents the sequence red, green.

```
thesis_state_2 : function() {
     this.playerVM.DisplayText("Hello World!");
     this.state = this.playerVM.SequenceButtonPress(["12"], [states.thesis_state_3]);
},
```

Figure 21 : A state function that displays text and encodes a colored sequence transition event

In this snippet, we see a KeyboardInput call being made to the VM. This functions exactly like a sequence button press except takes an array of input strings instead of button presses.

```
thesis_state_2 : function() {
     this.playerVM.DisplayText("Hello World!");
     this.state = this.playerVM.KeyboardInput(["input text"], [states.thesis_state_3]);
},
```

Figure 22 : A state function that displays text and encodes a text input transition event

In this last snippet below, we can see that the state is requesting to be transitioned directly to the next one without any type of transition.

```
thesis_state_2 : function() {
     this.playerVM.DisplayText("Hello World!");
     this.state = states.thesis_state_3;
},
```

Figure 23 : A state function that displays text and encodes an immediate transition to a state

Transpiler

After the template for the transpiler was done being designed, it was time to design the Java code whose responsibility is to take the data from the data model and transpile it to this JavaScript template. Due to the fact that the transpiler template is split up into parts, it made sense to also split the transpilers code up into different classes responsible for these different parts.

The transpiler functions by executing a list of ITranspilerSteps. Each ITranspilerStep has a method called PerformStep that must be implemented by any class who implements the ITranspilerStep interface. The transpiler looks through this list and calls each steps PerformStep method. The steps that have been implemented in the transpiler correspond one to one to the different parts of the transpiler template. For example, there is a GenerateNameSpaceAndVariablesStep whose responsibility is to generate the JavaScript of that part of the transpiler template.

```
var FSMGame = {

   gameInstanceId : 0,
   team : 0,
   player : 0,
   playerVM : null,

   running : true,
   state : states.thesis_start,
   oldState : null,
```

Figure 24 : The Transpiler's GenerateNameSpaceAndVariablesStep

After all of the transpiler steps are completed, all of the results from each of the steps are appended into a single JavaScript file and saved into the game servers program directory for execution.

Game Server

The game server is another vital component to the WLCP. It handles all of the communication between clients as well as physically runs the transpiled JavaScript games. It also handles server wide functions such as starting and stopping game instances as well as starting and stopping debug games instances.

Selected Tools

Like the other projects, the game server was developed using Java. The built in Java asynchronous tcp socket communication was also used. Other than that no other external dependencies were used other than the basic JPA and eclipselink libraries for communication with the database.

Design

The design of the game server could have gone two different ways. The first way was to use HTTP for communication between the game server and client. This would entail writing the game server as a SPRING HTTP controller. The advantage of this would be it would require much less development. The disadvantage is that it would be difficult to have complete control over communication and sending data to the client would be difficult without having first received a request. Push technology could be used, but it would be better to implement a communication protocol from scratch. This brings us to our second way which is to use asynchronous TCP communication with a custom communication protocol. This method was eventually chosen and it gave the most flexibility and complete control over everything. However, this was chosen knowing there would be significantly more development work.

When designing the game server, a few standards were defined that the server had to meet. It had to be heavily multithreaded and easily extensible. Both of these allow for scalability as the number of users increase. Rather than having all of the games and users run on a single thread and potentially a single CPU core, each game runs on its own thread as well as each player, meaning they can be spread out across cores and as core count increases so does the number of players the server can handle. Since game instances and players all run on their own thread this also means that it doesn't matter what server they are actually running on. A load balancer can be placed in front of many instances of the game server and redirect data to the correct one due to the design of the communication protocol. Since the server is multithreaded it was also chosen to use asynchronous tcp communication rather than synchronous communication to the multithreaded nature.

Modules

The design of the game server follows the module design pattern in software engineering. This means that the entire server is split up into different modules (usually singleton classes), that each provide their own separate functionality. In the game server, there are 4 different modules: the configuration, game server, logger and task manager modules.

The configuration modules provides a means for the host of the server to configure it to suit their needs. It does this by reading some config xml files located in the configuration directory. This allows for configuring the server host name as well as port number. It also allows one to configure the heartbeat (timeout) timer for clients.

The game server module is responsible for setting up all of the asynchronous tcp socket communication. This includes binding to an address, accepting incoming connections, and reading and writing to the channel. It is also responsible for handling websockets. This includes websocket hand shakes, connecting, disconnecting and reading and writing to the channel. Websockets have to be handled specially because they have a special header on top of all data transmitted through them. When data is read through a regular socket or websocket, it is then passed onto the packet distributor, which will distribute it to the correct location, a server task or a game instance task.

The logging module is responsible for writing output to the log console and log file. This includes all system output as well as any user configuration output via the logging API it provides.

The task manager module is responsible for managing running tasks. This includes starting and stopping them. Tasks will be described in the next section.

Tasks

A task is simply a thread. A task can be started by adding it to the task manager module and stopped by removing it. When started a task will run on a separate thread, so it is important to be aware of proper memory access techniques. The game server has 3 different tasks: game instance task, packet distributor task and server packet handler task.

When a user requests a game instance to be started from the web ui, a new game instance task is started. This means that all game instances run on their own thread. The game instance task is responsible for handling all game instance packets. These will be routed to the appropriate game instance via the packet distributor based on the game instance id in the packet header. Once they are routed they are then handled appropriately by the game instance task. This includes things such as users connecting and disconnecting, heartbeats and input transitions.

The packet distributor task is responsible for routing packets of different types to their appropriate tasks. There are two different types of packets a server packet and a game instance packet. When a server packet is received it is routed to the server packet handler (explained next) and when a game instance packet is received it is routed to the game instance that corresponds to the game instance id in the packets header (explained above).

The server packet handler task is responsible for handling server tasks and packets. This includes things such as starting and stopping regular and debug game instances. It also includes getting list of available game lobbies for a player to join.

VM

When a player joins a game instance, a player vm is started on a new thread. The player vm is responsible for configuring and starting the transpiled JavaScript game using the Nashorn JavaScript Engine. It configures some of the transpiled games variables like the game instance id, team number, player number and a reference to the Java player vm. It then invokes the start method in the transpiled code and the game begins for that player.

The transpiled JavaScript was passed a reference to the Java player vm meaning it can make calls to the player vm Java code. This beneficial because this means its possible call java methods to send and receive packets. This is done through the PlayerVM API which includes methods to send and receive Display Text, Single Button Press, Sequence Button Press and Keyboard Input packets. Below you can see how the transpiled JavaScript code maps to the PlayerVM Java code.

```
    thesis_state_2 : function() {
         this.playerVM.DisplayText("Hello World!");
    }
```

```
         public void DisplayText(String text) {
          gameInstanceTask.getPacketDistributor().AddPacketToSend(new DisplayTextPacket(text),
usernameClientData.clientData);
    }
```

Figure 25 : How the transpiled JavaScript code maps to the PlayerVM Java code

Here you can see that the transpiled javascript this.playerVM.DisplayText actually sends a new DisplayTextPacket to that client, so their device can display the text in the transpiled JavaScript. Below we will look at the more complex SingleButtonPress.

```
   thesis_state_2 : function() {
        this.playerVM.DisplayText("Hello World!");
        this.state = this.playerVM.SingleButtonPress(["1", "2"], [states.thesis_state_3,
states.thesis_state_3]);
   }
```

```
        public int SingleButtonPress(String[] buttons, int[] transitions) throws ScriptException {
         block = true;
         gameInstanceTask.getPacketDistributor().AddPacketToSend(new
SingleButtonPressPacket(gameInstanceTask.getGameInstanceId(), team, player, 0),
usernameClientData.clientData);
         int state;
         while((state = block()) == -2) {}
         if(state != -2 && state != -1) { return state; }
         SingleButtonPressPacket packet = null;
         if(blockPacket instanceof SingleButtonPressPacket) {
                 packet = (SingleButtonPressPacket) blockPacket;
         } else {
                 return gotoSameState();
         }
         for(int i = 0; i < buttons.length; i++) {
                 if(buttons[i].equals(Integer.toString(packet.getButtonPress()))) {
                         return transitions[i];
                 }
         }
         return gotoSameState();
    }
```

Figure 26 : How the transpiled JavaScript code maps a Single Button Press

For transitions things are a little bit different. They have to implement blocking. First they send a Single Button Press Packet to the device to tell it to block for input and what type of input (single, sequence, keyboard, etc). Then the player vm blocks until it gets a response back from the client. When it does, it returns the state number to transition to depending on how the transition was configured. Sequence button press and keyboard input work in very similar ways and won't be gone into detail about.

Game Server API

The game server API is a very basic Java API meant for people who want to develop a Java client for the WLCP. It provides a means of connecting, disconnecting, sending data and receiving data to and from the server. This API masks a lot of the backend setup and communication with the server making it very easy to implement new clients on any devices that run Java and have an internet connection such as Android devices.

Shared Library

The shared library project contains class definitions that need to be used when communicating with the game server. These class definitions define the packet structure that is to be used when communicating with the server. In this case, the library is written in Java meaning this library should be used when communicating to the server in a Java environment. For example, writing an Android player client.

All packets must extend a base type Packet and implement the interface IPacket. This ensures that they implement required methods such as populateData and assemblePacket which are necessary for reading and writing data from and to packets. The base type Packet only contains two fields, packetType (enum byte) and packetSize(int).

The library contains two packet definitions called GamePacket and ServerPacket that extend Packet. These two classes add on a few different fields to Packet depending on which one. GamePacket adds on gameInstanceId (int), team (int) and player (int). These fields are used to make sure the packet makes it to the correct game instance and player. This class is meant to be extended when creating packets that go to and from a game instance on the server. ServerPacket doesn't add any fields, but is meant to be extended when creating a packet that is meant for the server in general and not game specific such as starting a new game instance.

Study

In order to prove if the use of the WLCP helps with computational thinking skills and allows for easy creation of multiplayer embodied games, a study was performed. The study involved around 15-20 participants and were broken up into groups over several days in which participants designed games on paper, entered them into the WLCP, debugged and played. After the study, the games would be coded using the Multiplayer Game Design and Computational Thinking Coding Guide by the Embodied Games Group at WPI.

Participants

Worcester Polytechnic Institute and the Embodied Games Group at WPI have a great relationship with The Massachusetts Academy of Math and Science. This school is a free, co-educational, public school of excellence that enrolls approximately 100 academically accelerated 11th and 12th grade students from Massachusetts. Math and science are emphasized within a comprehensive, interactive academic program. Rigorous junior year classes and senior classes taken at Worcester Polytechnic Institute (WPI) prepare students for college academics before they've graduated from high school (Massachusetts Academy of Math & Science at WPI).

Since the platform was still in its early stages we decided to test with older students. At the time of this writing, studies have already been done with younger students and these results will be published elsewhere.

Fortunately Mass Academy was able to provide us with 22 students to participate in the study as an elective program as the students were required by the school participate in an after school elective. Of the 22 participants, 18 of them showed up for the after school program. These 18 participants were broken up into 6 teams of either 3 or 4 people per team.

For each team a Google Drive folder was created for them to store any documents or deliverables they were required to complete. Each team folder would also contain a folder for each individual of the team. This structure would be used to store the results for both teams and team members.

Procedure

The study was run as an after school program at The Massachusetts Academy of Math and Science. It ran for 1 hour from 3-4 PM every tuesday for the next 6 weeks (total of 6 days). At the beginning of each day a presentation would be given to give the participants some background on the days tasks and goals. The following sections contain these activities, their description and their purpose.

Day 1 : Playing EstimateIt!

The first day of the study was meant as an introduction to what the participants would be doing over the next 6 weeks.

Presentation

At the beginning of the first day, a presentation was given to the students on how the rest of the elective would be run. The group was split into half for the day. Since each day was 1 hour long, the day was split up into two 30 minute blocks. Each half of the participants would participate in a 30 minute block and then switch. By the end of the day both groups will have played EstimateIt! and taken the pre test. The presentation also outlined how to access the online pre study as well as play EstimateIt!

Activity

Pre-Study

The first block was taking a pre-study questionnaire. Participants would spend 30 minutes filling this out and if they didn't finish, they could take up to 30 minutes for homework to finish as much as they can since each student was assigned a google drive folder. They weren't required to finish the entire thing (due to time constraints). The purpose of this was to get a metric of their understanding of finite state machines at the beginning of the study.

EstimateIt!

The second block consisted of playing the game EstimateIt! This game is a scavenger hunt game where players have to hunt for different objects by estimating their size. The game is split up into 3 teams with 3 players per team. In level 1, players work individually to find objects but then in level 2 and 3, players work together as teams to find objects. The purpose of playing this game was to get the participants familiar with the types of games they would be creating the remainder of the elective.

Homework

The participants were asked to take up to 30 minutes for homework to finish filling out as much as they could of the prestudy they started on earlier in the day.

Day 2 & 3: Game Creation

Days 2 and 3 of the study were dedicated to designing a math game with the other people assigned to your team.

Presentation

At the beginning of day 2, a presentation was given to the participants before the activity that day. The presentation contained one slide that outlined how students would be broken up into 6 groups of 3-4 partners. It contained participants names and the team they have been assigned to. These teams would be used for the rest of the study.

At the beginning of day 3, a presentation was given to the participants on what a Finite State Machine is (FSM). This presentation explained what a state is, what a transition is and what a state machine is. It also showed some examples of FSMs. The presentation also explained how EstimateIt! works behind the scenes and how it uses a FSM. By showing this, the WLCP was introduced to the participants. Lastly, it contained instructions on converting the games they designed on paper on day 2 into FSMs.

Activity

On day 2, the participants were told to put themselves in the shoes of a younger student (3rd or 4th grade) when designing the game. They were given a variety of writing utensils as well as a large multipage writing pad. Students were then told to design a math game, describe the game and draw a representation on the paper pads. The game had to meet the following criteria:

- A game that 3-4th grade kids can play in school over recess (it can be played in the classroom or outside in a playground/park or in the gym),
- The game has to teach (or allow students to practice) a particular math concept
- The game should have at most 4 players and if there are teams, at most 2 teams
- We want to get the students moving, so the game must be active; it should require physical movement by the students
- Ideally, the movement should be connected to the math in some way.

- The game should involve mobile technology (cell phones)
- We want you to specify the game (show us how your game works) on these pads on paper, and we will ask you to explain how it works later.

At the end of class if there was time, students would give a small 3 minute presentation on the game they designed to the rest of the class.

On day 3, the students were tasked with converting their games from day 2 into FSMs, now that they had a presentation on what a FSM is. They would perform this on their paper pads that they used from day 2.

Homework

The homework was the same for days 2 and 3. This was to upload a document to their teams google drive folder containing the following:

- Please write instructions that describes how your game works. This should be written as if you were explaining your game to 4th-6th grade students (2 pages).
- Please draw a representation (pictures/diagrams) that demonstrates how your game works. *(You can take a photograph of your pads to add to the report if you want)*
- Any changes made to the game.

Participants were told to take up to 30 minutes on this assignment.

Day 4 & 5 : Wearable Learning Cloud Platform

Presentation

On day 4 a presentation was given to the participants. This presentation was a tutorial on how to use The Wearable Learning Cloud Platform. This is the tool suite that they would be using for the remainder of the study. This would allow participants to create and edit games as well as play and debug them. It was explained how the games could be multiplayer games and examples were demonstrated to the participants.

Activity

Now that students have been introduced to The Wearable Learning Cloud Platform, they were tasked with converting their paper FSM games into WLCP FSM games. This would include using the WLCP Game Editor which would allow students to define their states, connections and transitions and then debug and play them. Since students had many questions during these two days of the study, extra staff was on hands to help answer particiaptns questions promptly and thoroughly.

Homework

Day 4 and 5 homework was similar to days 2 and 3 homework. Students were asked to spend up to 30 minutes each night documenting their ideas and progress.

Day 6 : Testing and Debugging

Presentation

At the beginning of the day, participants were instructed on how the day would organized.

Activity

The first 15 minutes of day was allocated for teams finalizing their games for play testing. Once the 15 minutes was up each team was given about 5 minutes to present their game and and pick some volunteers from the rest of the participants to play it. The rest would spectate. This would continue for each team until all days were presented and played. This allowed for everyone to see everyone elses game and give them a chance to play some. The last 10 minutes of the day would be allowed to filling out a small survey.

Homework

The first homework consisted of filling out a post-study questionnaire. Participants were instructed to take up to 1 hour to complete this.

The second homework consisted of documenting how play testing went. Participants were instructed to take up to 10 minutes to complete this and were asked to answer questions such as:

- How did playtesting your game go?
- Did you find any bugs in your game?
- What would you have changed about your game?

RESULTS

Measures

In order to answer my study questions proposed in the beginning of this thesis write up, I would need to be able to measure some type of quantifiable data. In order to do this, data would have to be captured before the study, during the study and after the study. As explained in the study section this would be done by proctoring a pre and post test, survey and keeping all assets and game products (that the participants generated) after the study was completed to be later coded.

Pre & Post Test

The pre and post test was given to the students on the first and last day respectively. Students were given about 30 minutes for both and told they can spend extra time at home completing it. The pre and post test contained the same questions. By having it taken before the study and after the study, one could quantify if the participants learned anything. The pre test would only contain the average amount of questions completed during the post test. This was done because one could not quantify if a participant learned something if there is no post data to compare the pre data to.

Out of the 18 participants in the study, 18 completed the pretest and 9 completed the post test on computational thinking. All of the students completed the pretest because it was required to be done in class. The post test was assigned as homework on the last day of the study, so there was no guarantee every participant would complete it. Of the 9 participants that completed the post study, an average of the first 3 questions were completed. These 3 questions can be found in Appendix **A.** The table below shows a summary of the pre and post test.

Table 1 : Means, Standard Deviations and T-test results for Gains in Computational Thinking

Paired Samples T-Test						
	M	SD	n	95% Confidence Interval for Mean Difference	t	p
Posttest-Pretest	0.17	0.18	9	0.04, 0.31	2.91	0.02*
Post Q1 - Pre Q1	0.13	0.27	9	-0.07, 0.34	1.51	0.17
Post Q2 - Pre Q2	-0.37	0.14	9	-0.14, -0.07	-0.80	0.45
Post Q3 - Pre Q3	0.44	0.55	9	0.02, 0.87	2.41	0.04*
						*Significance level at *p<0.05*

As you can see, only the first 3 questions of the pre and post test were used. Looking at the overall statistics, students improved by 17% from the pretest to the posttest, and this difference is statistically significant ($t(9)=2.91$, $p<0.05$). When analyzing questions individually, we see that students improved most in question 3 (creating a finite state machine from scratch), and the difference for question 3 is significant, while the improvement in questions 1 (interpreting a state machine) and 2 (modifying a state machine) showed no significant improvement from pre to posttest. This is mostly likely due to the fact that the study was focused around interpreting and creating finite state machines and not modifying them. If we take a look at just questions 1 and 3 we can see that the students improved by 28.5%. This is a pretty significant increase and answers the research question whether a person's knowledge of Finite State Machines change by designing games on paper and in the WLCP.

Post Study Survey

On the last day of the study, participants were asked to take the last 10 minutes of that day to fill out a post study survey. If they didn't finish it during the last 10 minutes, they were asked to try and finish it up for homework. This survey mostly concentrated on questions that have to do with using the WLCP. The purpose of this survey was to get feedback from the participants on this such as how easy the WLCP was to use, any difficulties they faced, any features they wish it had, etc. Out of the 18 participants, 14 completed the survey. This survey can be found in **Appendix B.**

The first question on the survey asked what participants **liked** about the WLCP. Most participants liked how “user friendly” it was and how “it's simple and easy to understand.” Others liked its aesthetic, the drag and drop interface and how “little programming knowledge was needed.” This proves that it is possible to create a user friendly and easily usable of designing and playing finite state machine games digitally.

The second question on the survey asked what participants **disliked** about the WLCP. Most participants disliked that there was no way to represent variables or expressions. Also that there was limited functionality (only display text, button presses) and some bugs made it very frustrating at points.

The third question on the survey asked participants what features they would like to see added to the WLCP. Participants wanted to see the addition of variables, expressions, communication with other devices, QR codes, GPS, and other types of new sensors. Some also wanted things such as copy and paste features.

Skipping to the ninth question, participants were asked how hard it was to transfer their games from paper to the WLCP. The question was based on a 1 to 5 scale with 1 being very easy and 5 being very hard. The results are shown below.

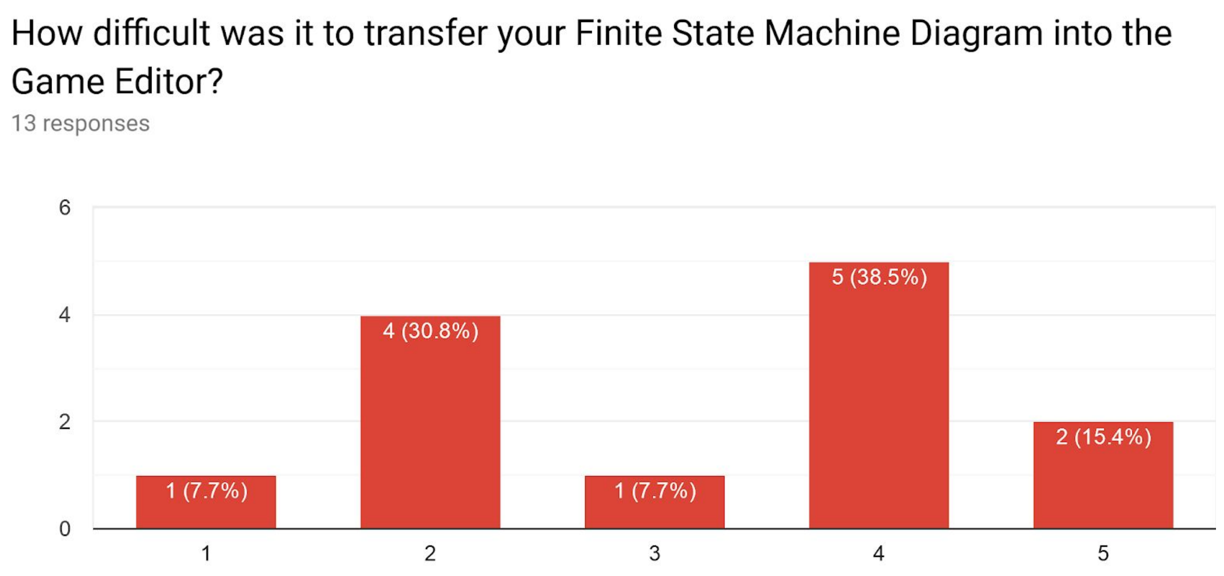


Figure 27 : Student Survey Responses: Ease of transferring the FSMD from paper to the WLCP

As you can see, 5 people found it easy and 7 people found it hard and 1 person found it intermediate. The next question asked to justify their answer using text. The participants who found it hard to transfer over said it was because the editor did not support features they needed. The participants who said it was easy stated that they had to make some modifications to their games, but once they did, the transfer wasn't that hard.

The past few questions answer the research question about it users would encounter limitations when converting their FSMD games into WLCP FSMD games. Users did encounter limitations when transferring their games. This however was mostly due to the WLCP lacking features or having bugs causing users not to be able to do things. With more time to develop the WLCP, users would encounter less limitations.

Skipping to the 14 and 15 question, participants were asked if they think younger students would be able to use the game editor. Below is the pie chart of the responses from question 14.

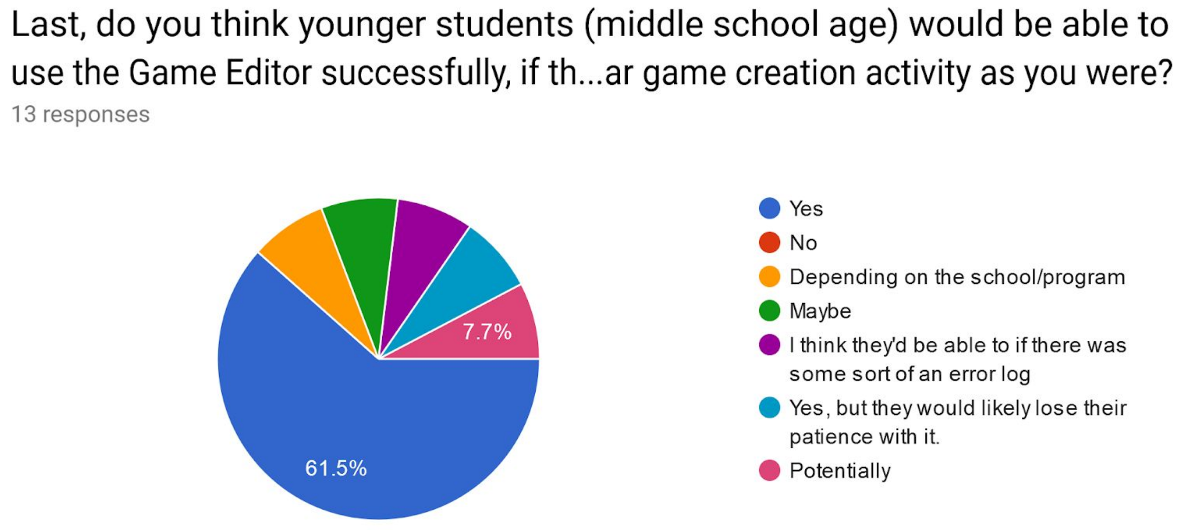


Figure 28 : Student Survey Responses: Perceptions of Ease of use of Game Editor

The response was an overwhelming yes with no participants responding an explicit no. This question allowed for other responses. By addressing the issues in those responses, it is with no doubt this could be implemented with younger students. Question 15 asked participants to justify the answer to the pie chart. Most responses were as follows : "it is pretty easy to use, instructions clear", "It was self explanatory and easy", "the editor was really easy to us." Other responded saying things such as : "needs error logging", "if certain bugs were fixed" and "if more features were added." Just like the pie chart, these are minor things and can be easily addressed and fixed. This answers another research question, if the WCLP is suitable for use by younger students.

Game Products & Coding

While students were creating their games, they were asked to work out of large sketch pads that were provided to them. They were also asked each day for homework to take pictures of all of their work and writeup what they did into a Google Document. This was done so that the Multiplayer Game Design and Computational Thinking Coding Guide could be used to determine things such as the types of game students created and how well their finite state machine diagrams held up to a standard.

Coding Guide

The coding guide that used to code the game designed and created during the study is broken up into many sections. A diagram of these different sections is shown below.

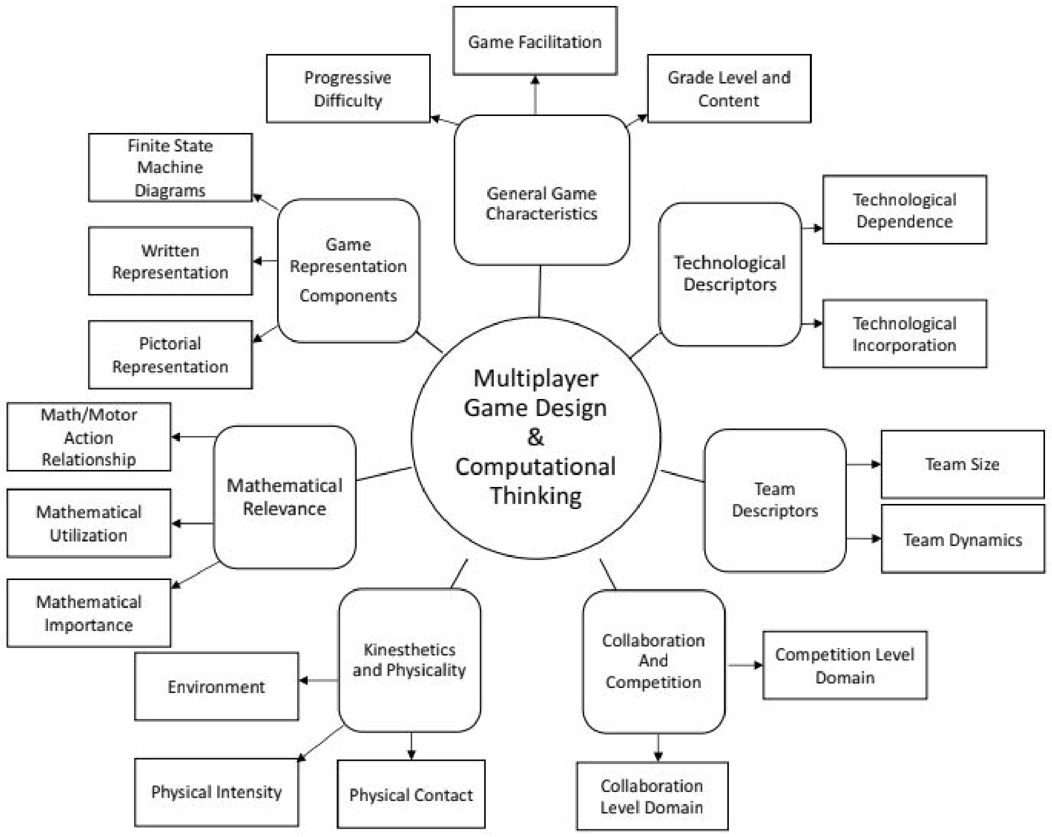


Figure 29 : A diagram of the Coding guide for Multiplayer Game Design and Computational Thinking, created by the Embodied Games group at WPI

Because this thesis is concentrated mostly towards creation of games using the WLCP, only select sections from this will be used to code the data. This will include Finite State Machine Diagrams section and other relevants ones to the WLCP such and collaboration and competitions, team descriptors, technological descriptors, etc.

Coding

The coding was done by three students in the WPI Embodied Research Group. All three students were properly trained before coding and have had prior experience using this guide to code other data. After students have completed coding by using the coding and data provided, inter reliability checks were done between students to verify that coders were agreeing on things. If coders don't agree within a certain percent the results could be considered questionable. The three students had 86% agreeability. Anything above 80% is considered to be high agreeability and thus highly reliable data.

Finite State Machine Diagram Coding

Day 3

After day 3 and 5 of the study were coded, the results could be compared. It was decided to compare day 3 and 5 FSMDs because we wanted to compare their on paper FSMDs to digital FSMDs.

Five of the six teams (83%) created a diagram on Day 3, the results of which are highlighted below in **Table 1**. The FSMD Elements marked the presence of input types, the domain level of the FSMD, and the presence of FSMD features such as If-Then Statements that would suggest Evidence of Programming Language Knowledge (EPLK). The FSMD Criteria regarding representation, consistency, and completion were coded on a 4-point Likert scale based on the overall diagram (0=Never, or mostly never meeting criteria; 3=Always, or Almost Always meeting criteria) while the FSMD Quantitative Analysis items were coded on a continuous scale to count the individual components of the FSMD

Table 2 : Means, Standard Deviations and Frequencies of Technology Elements included by students in the Finite State Machine Diagram Representations on paper (day 3)

FSMD Elements	**N**	**Percent**	**FSMD Criteria**	**Mean**	**SD**	**MODE**
Input Types : RFID	0	0	Output State Representation	2.6	0.89	3
Input Types : Buttons	4	0.8	Transition State Representation	3	0	3
Input Types : GPS	0	0	Consistency with Rules	2.8	0.44	3
Input Types : Keyboard	4	0.8	State Consistency	2.6	0.54	3
Input Types : Touch Interface	1	0.2	Transition Consistency	2.4	0.89	3
Input Types : Timer	0	0	Completion	2.8	0.44	3

Input Types : Another Player	1	0.2	**FSDM Quantitative Analysis**	**Mean**	**SD**	**Mode**
Input Types : Other (please write)	0	0	States / Boxes	7	3.54	NA
Domain Level : Management-Level	1	0.2	Transitions / Arrows	9.8	4.60	NA
Domain Level : Team-Level	3	0.6	Labeled Arrows	7.8	5.06	NA
Domain Level : Player-Level	3	0.6	Numbered States	0	0	0
EPLK : If-then Statements	3	0.6				
EPLK : Programmatic Loops	4	0.8				
EPLK : Loop to Prior State Arrows	4	0.8				

Looking at the coded results we can see that for FSMD Elements most teams were using button presses or keyboard input for their input types. Most games were also defined at the team or player level, meaning they were mostly multiplayer oriented. The games also contained a decent amount of EPLK. Looking at the FSMD Criteria, we can see the averages are pretty close to 3 (perfect representation), but because the diagrams are being done by hand on paper there is still some human error. Lastly, looking at FSMD Quantitative Analysis we can see that the teams games on paper weren't very complex due to the little number of states and transitions they contained.

Day 5

The next day that was coded was day 5. On this day students were using the WLCP to create their Finite State Machine Diagrams rather then drawing them in their pads. Below are the coded results for the WLCP Finite State Machine Diagrams from day 5.

Table 3 : Means, Standard Deviations and Frequencies of Technology Elements included by students in the Finite State Machine Diagrams in the WLCP Game Editor (day 5)

FSMD Elements	**N**	**Percent**	**FSMD Criteria**	**Mean**	**SD**	**MODE**
Input Types : RFID	0	0	Output State Representation	3	0	3
Input Types : Buttons	5	100	Transition State Representation	3	0	3
Input Types : GPS	0	0	Consistency with Rules	2.6	0.54	3

Input Types : Keyboard	3	0.60	State Consistency	3	0	3
Input Types : Touch Interface	0	0	Transition Consistency	3	0	3
Input Types : Timer	0	0	Completion	2.2	0.44	2
Input Types : Another Player	0	0	**FSDM Quantitative Analysis**	**Mean**	**SD**	**Mode**
Input Types : Other (please write)	0	0	States / Boxes	43	51.45	NA
Domain Level : Management-Level	1	0.20	Transitions / Arrows	55.2	69.12	NA
Domain Level : Team-Level	3	0.60	Labeled Arrows	54.6	68.85	NA
Domain Level : Player-Level	3	0.60	Numbered States	43	51.45	NA
EPLK : If-then Statements	3	0.60				
EPLK : Programmatic Loops	4	0.80				
EPLK : Loop to Prior State Arrows	4	0.80				

Looking at the coded results we can see that for FSMD Elements teams were only using button presses or keyboard input. This is because at the time of the study the WLCP only supported those types of inputs. Most games were also defined at the team or player level, meaning they were mostly multiplayer oriented. The games also contained a decent amount of EPLK. Looking at the FSMD Criteria, we can see the averages are all 3 except for Consistency With Rules. Since the participants were using the editor it wasn't possible to score lower than 3, because you were forced to use its representation which is considered the correct way. Lastly, looking at FSMD Quantitative Analysis we can see that the teams games in the WLCP were very complex and boasted a very large amount of states and transitions. This is because by creating these digitally they had as much space as they needed and the drag and drop interface made it very easy to create and connect things.

Day 3 & 5

We can compare the results from day 3 and 5 to get an idea of things such as how the participants games changed when going from paper to the WLCP.

The input types do not differ much, with the exception of Touch Interface and Another player on day 3. This is because on day 3 we did not tell them that the WLCP was restricted to only button presses (single, sequence) and keyboard input. As you can see on day 5, all teams were using button presses or keyboard input because of this restriction.

Looking at Domain Level, we can see that the results are exactly the same for day 3 and 5. We can see that all of the teams created games that mostly use the team and player level management. This is because participants were instructed in the beginning of the study to design a multiplayer game.

The FSMD criteria determines how well their diagrams held up to a standard. The paper games scored lower than the WLCP games. This is because the paper games contain human error abiding to the standard, where when using the WLCP you automatically abide by the standard.

Lastly, comparing the FSMD Quantitative Analysis, we can see that the WLCP games have significantly more states, transitions and connections. Almost 500% more in some cases. This is because the WLCP allows you to create games much faster than on paper. This is due to its drag and drop interface, validation engine, user friendliness and rapid prototyping and debugging.

In general, we can say that the FSMD representations mostly changed in the FSMD representation accuracy and amount of states and transitions. It also mostly remained the same in the types of inputs and outputs they used.

Game Products

At the end of the study, we had a huge collection of data and this can be called the game products. This consists of games on paper in each teams sketch pads (as well as google drive), homeworks for each night (google drive), finite state machines on paper and also their games they created using the WLCP. Below are some examples of these game products.

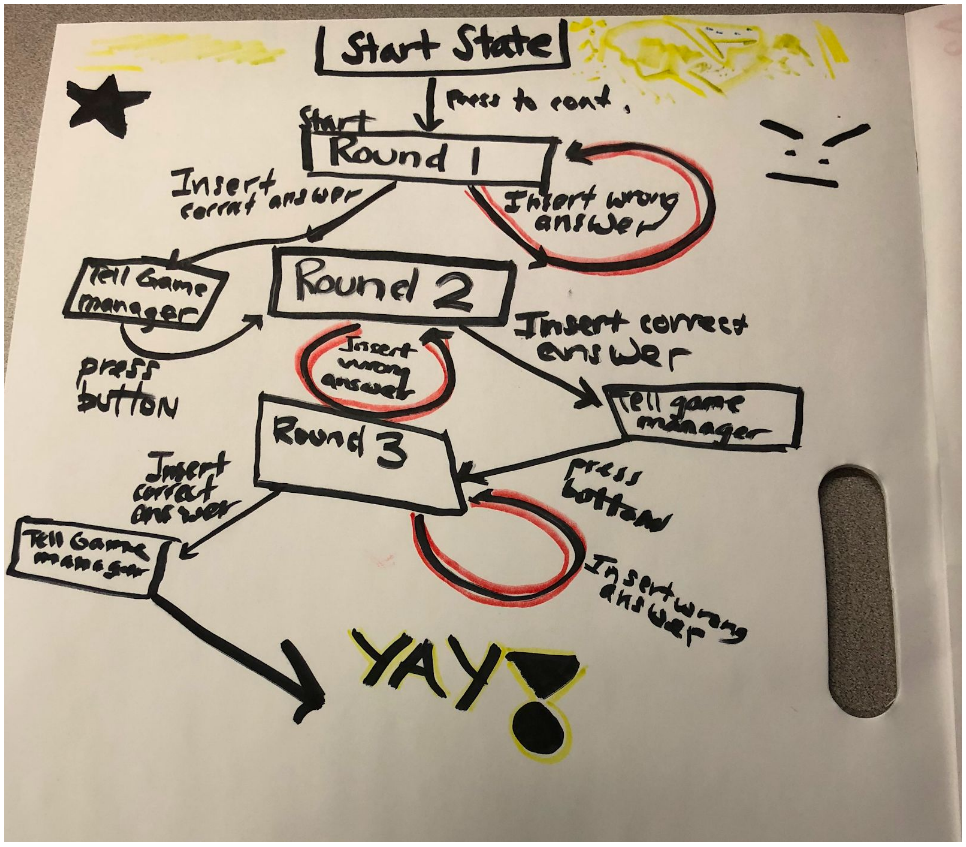


Figure 30 : Paper FSMD Game

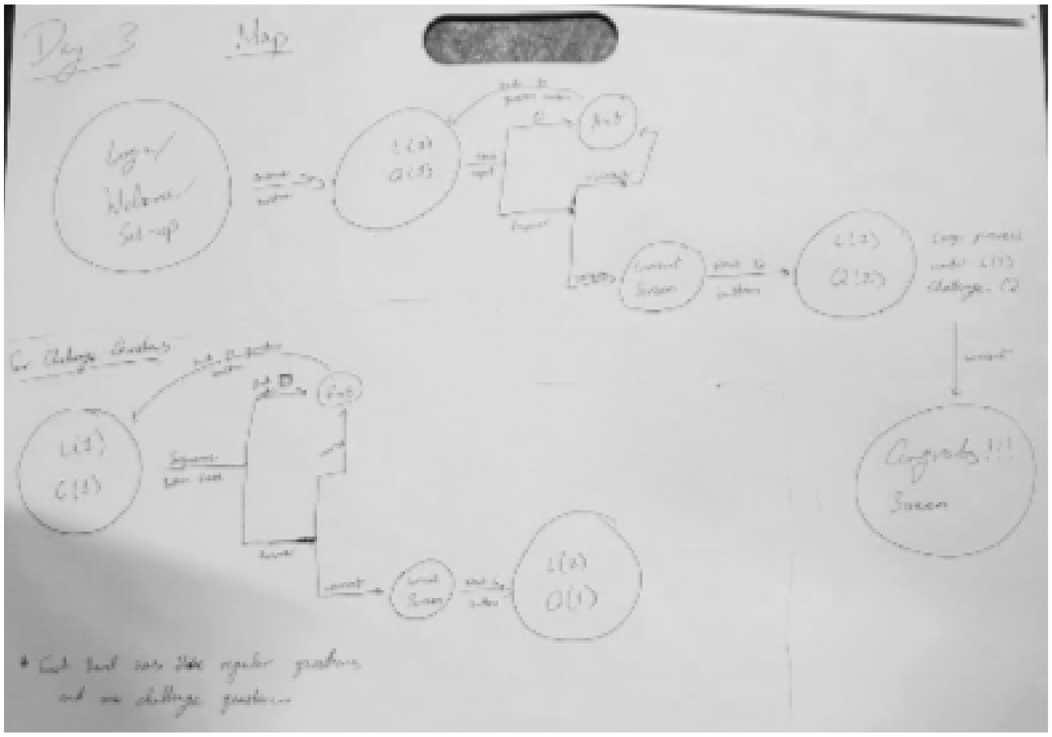


Figure 31 : Paper FSMD Game

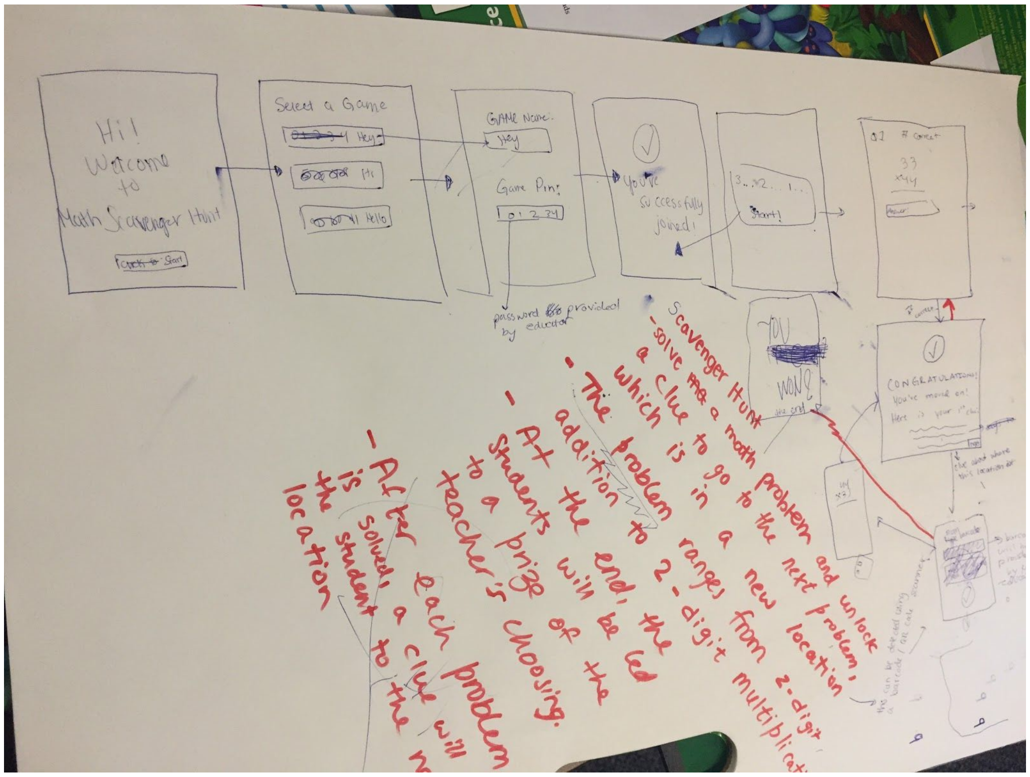

Hi! Welcome to Math Scavenger Hunt
Start
Select a Game
Hey
Hi
Hello
GAME Name:
Hey
Game Pin:
0123,24
password provided by educator
You've successfully joined!
3..2...1...
Start!
YOU WON!
CONGRATULATIONS! You're moving on!
Scavenger Hunt
- solve a math problem and unlock a clue to go to the next problem, which is in a new location.
- The problem ranges from 2-digit addition to 2-digit multiplicat
- At the end, the students will be led to a prize of the teacher's choosing.
- After each problem is solved, a clue will the student to the n
location

Figure 32 : Paper FSMD Game

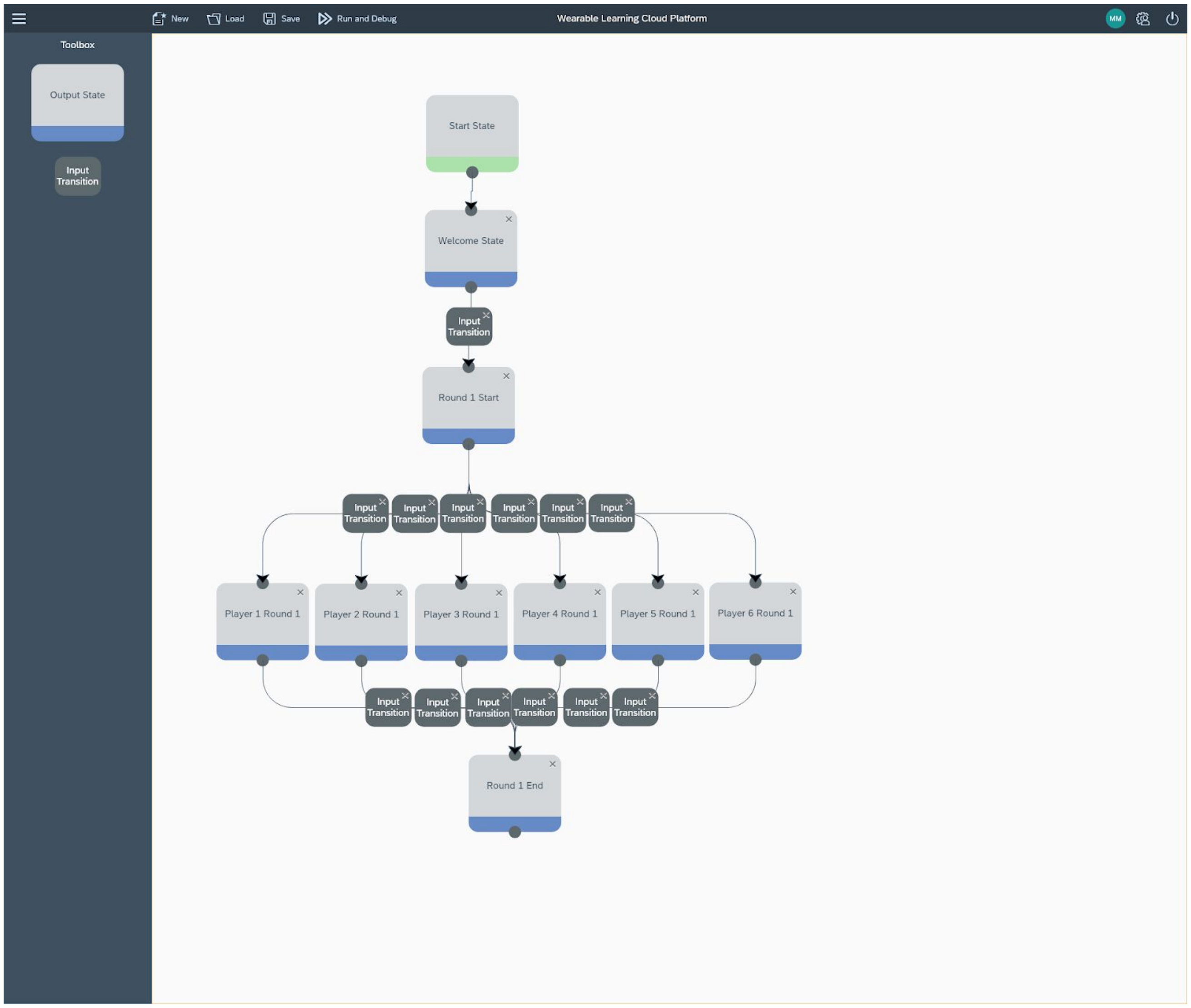


Figure 33 : WLCP FSMD Game

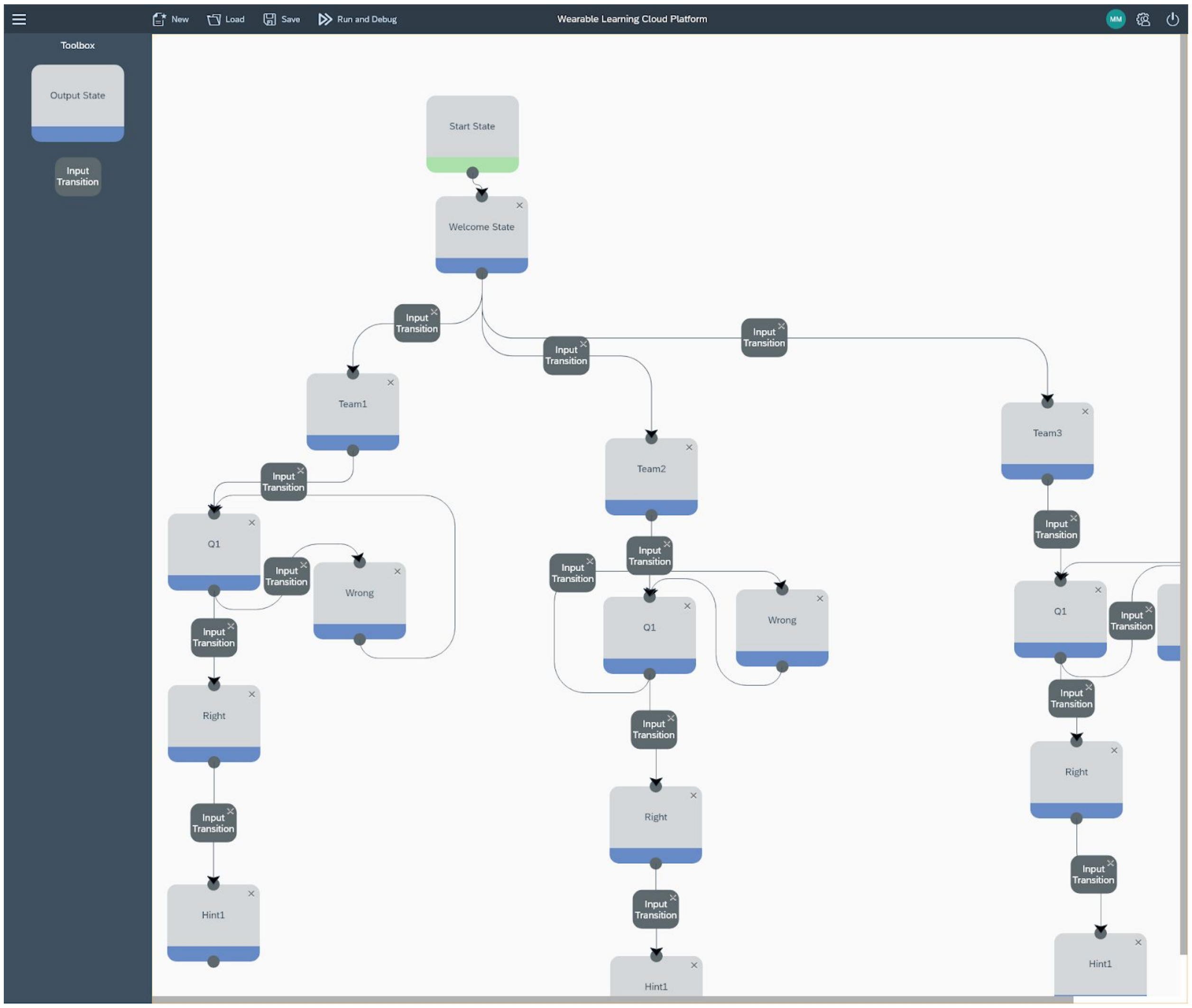


Figure 34 : WLCP FSMD Game

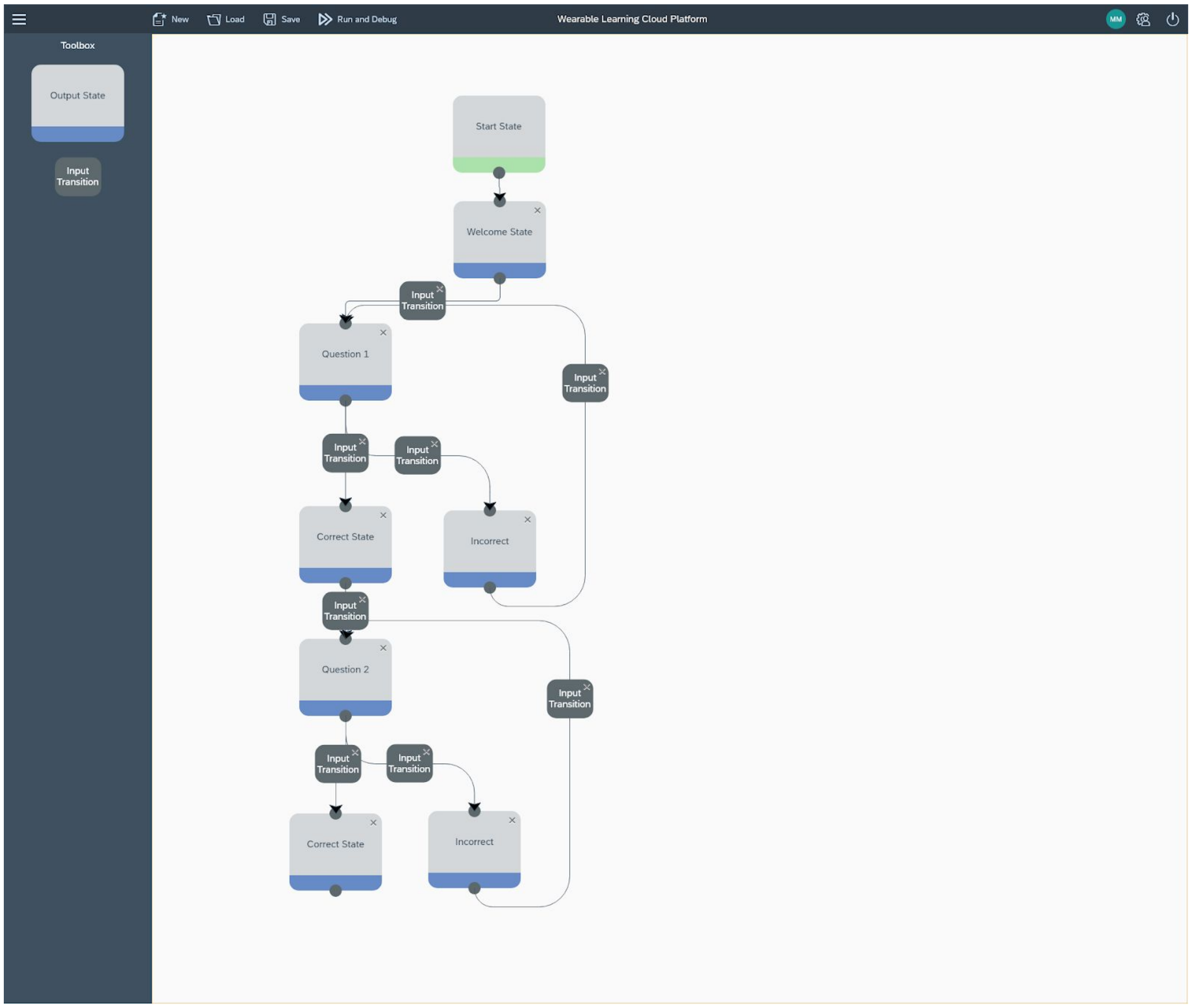


Figure 35 : WLCP FSMD Game

DISCUSSION

This thesis started with the ambitious goal of creating a programming language and software environment for the creation of multiplayer, physically active educational games. I consider this goal was accomplished and that each of the research questions have been answered to a reasonable extent. I will now revisit each research question proposed in the beginning of this report and summarize if the question has been answered or not.

- **Regarding Student’s Computational Thinking**
    - Does a person's knowledge of defining a computational system *via* Finite State Machines (one way to assess students’ Computational Thinking ability) change by designing multiplayer math games (both on paper measures and in the WLCP)?

The results from the pre and post test were used to determine the answer to this question. According to the results, students had and overall increase of 17% from the pretest to the post test. If you exclude Q2 (study did not focus on modifying diagrams) then the participants had an overall increase in 28%. This shows the students gained knowledge on FSMDs.

- **Regarding the visual programming language**
    - Is the WLCP suitable for use by K-12 students?
    - Is it possible to create a user-friendly mechanism to design and play multiplayer ubiquitous games (defining these finite state machine-based games digitally)?

The results from the post study survey was used in determining the answers to the research questions above. According to the survey, participants said the WLCP was user friendly and easy to use. They also agreed that they think even younger students would be able to use it (elementary and middle school). Participants were also able to successfully design and play multiplayer digital FSMD games using the system. Both of the questions above are answered with a strong yes.

- **Regarding Students' Coding and Game Development Processes**
    - How well can someone translate a game they designed on paper into a Finite State Machine Diagram on paper?
    - When transfering games from FSMD on paper to defining them on the computer (WLCP FSMD games), did users encounter limitations?

The answers to the research questions above were obtained by using the embodied coding guide and coding the games on paper from day 3 and the games in the WLCP from day 5. Since all students were able to translate their games from paper to the WLCP, this answers our first question. Games can translate very well from paper to the WLCP, however there can be some issues, but you can work around them. For our second question, users do encounter some issues when transferring their games. This is usually due to limited input and output types and not having variables or expression evaluation. Participants were still however able to work around this.

FUTURE WORK AND CONCLUSION

The WLCP will continued to be developed by the Embodied Games Research Group at WPI with consulting provided by me. New output types such as images, video and sounds will be added. New input types such as GPS, RFID, QR Code, another player will also be added. There will be overall improvements and bug fixes from things we found during the study that need improvement. During the study participants also recommended a copy and paste feature and even collaborative editing. These features may also be added down the road for later studies.

I consider this project to be a huge success. All of the research questions that were initially proposed in the beginning were answered. More excitingly, the answers to these questions were the ones I wanted to see. I wanted it to be possible to create a language and game editor for easy creation of embodied math games. This puts us one step closer to schools potentially creating entire curriculums about creating games. After working on projects with playing games, it was very interesting to see how one would turn out with creating games or if creating such a language to create these was even possible. This project has provided a solid foundation for The Embodied Research group at WPI to now continue to make improvements, add features and run more studies on game creation.

REFERENCES

Arroyo, I., Liu, Y., Wixon, N., Schultz, S. (2016) Toward Embodied Game-based Intelligent Tutoring Systems. Proceedings of the 13th International Conference on Intelligent Tutoring Systems. Zagreb, Croatia. Lecture Notes in Computer Science, Springer.

Arroyo, I., Schapira, A., Woolf, B. P. (2001) Authoring and sharing word problems in AWE (Animalwatch web-based environment). Proceedings of the Tenth International Conference on Artificial Intelligence in Education. San Antonio, TX. May 2001. pp. 527-529. IOS Press.

Arroyo, I., Micciollo, M., Casano, J., Ottmar, E., Hulse, T., Rodrigo, M. (2017) Wearable Learning: Multiplayer Embodied Games for Math Proceedings of the ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play (CHI PLAY). Amsterdam, Netherlands.

Casano, J., Arroyo, I., Agapito, J.L., Rodrigo, M.T. (2016) Wearable Learning Technologies for Math on SmartWatches: a feasibility study. In Chen, W. et al. (Eds.) (2016). Proceedings of the 24th International Conference on Computers in Education. India: Asia-Pacific Society for Computers in Education .

Computational Thinkers. 2018. Computational Thinking. https://www.computationalthinkers.com/wp-content/uploads/2016/01/ComputationalThinkingProductLogo-600x600.png

Ottmar, Arroyo, Castro, Hulse, Chatani, Harrison, Micciolo. (2017) MULTIPLAYER GAME DESIGN AND COMPUTATIONAL THINKING CODING GUIDE.

Habgood, M.P.J & Ainsworth, S.E. (2011) Motivating children to learn effectively: Exploring the value of intrinsic integration in educational games. Journal of the Learning Sciences 20 (2),169-206 .

Harrison, A., Hulse, T., Manzo, D., Micciolo, M., Ottmar, E., & Arroyo, I. (2018). Computational thinking through game creation in STEM classrooms. Proceedings (Part II) of the 19th International Conference on Artificial Intelligence in Education. London, U.K. pp. 134-138.

Hulse, T., Harrison, A., Micciolo, M., Arroyo, I., & Ottmar, E. : Engaging in computational thinking by creating math games. Paper submitted for the International Conference of the Learning Sciences, London, England (2018).

Hopcroft, J.E., Ullman, J.D. (1979) Introduction to Automata Theory, Languages, and Computation. Reading, MA: Addison-Wesley.

Lester, J.C., Ha, E.Y, Lee, S.Y., Mott, B.W., Rowe, J.P., Sabourin, J.L.: Serious games get smart: Intelligent game-based learning environments. AI Magazine 34 (4), 31-45 (2013).

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. The scratch programming language and environment. ACM Transactions on Computing Education (TOCE). 10(4).

Massachusetts Academy of Math & Science at WPI. https://www.massacademy.org.

Medium. 3 Things To Know About Scratch 3.0. 2018. https://medium.com/scratchteam-blog/3-things-to-know-about-scratch-3-0-18ee2f564278

Micciolo, M.: Physical Games for Learning II. Major Qualifying Project. Worcester Polytechnic Institute. (2017).

Price, T.W., Dong, Y., Lipovac, D.: iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In: ACM Technical Symposium on Computer Science Education. (2017).

Shute, V., Sun, C., Asbell-Clarke, J. (2017) Demystifying computational thinking, In Educational Research Review, Volume 22, Pages 142-158.

Steinkuehler, C., Squire, K., Barab, S. (2012) Games, Learning, and Society: Learning and Meaning in the Digital Age. Cambridge University Press .

Silver, E.A., Cai, J.: An Analysis of Arithmetic Problem Posing by Middle School Students□. Journal for Research in Mathematics Education, Vol. 27, No. 5 (Nov.), pp. 521- 539 (1996).

Tarnoff, B.: Tech's Push to Teach Coding Isn't About Kids' Success – It's About Cutting Wages. ACM Careers. (2017).

Van Horn, R.: Educational Games. The Phi Delta Kappan, 89(1), 73-74. (2007).

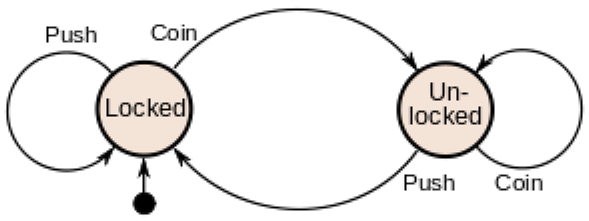
Wing, J.: Computational thinking. Communications of the ACM 49(3), 33-35 (2006).

Yazzie-Mintz, E.: Latest HSSSE results show familiar theme: bored, disconnected students want more from schools. UI News Room. (2010).

APPENDIX A. Pretest/Posttest in Computational Thinking

*Please answer the following questions to the best of your ability. To help with future research, please write an **X** beside any question that is unclear. Thank you!*

This is a visual model of how a machine works.

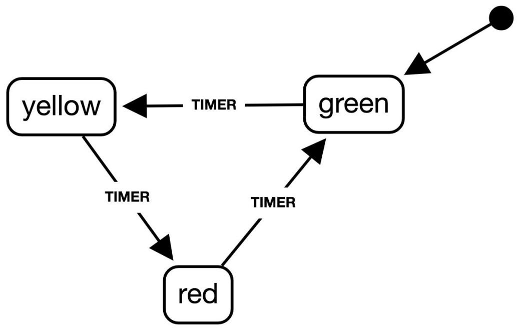


1a.) Can you tell what kind of machine this could be? What does it remind you of and why?

1b.) According to this model, what would happen if you put three (3) coins in? Can you?

1c.) The machine is currently locked. What would you have to do to get this machine to go "unlocked" and then back to "locked"?

This is a model of how a traffic light works. It changes lights based on a timer.



2a.) We want to add a button that a pedestrian could push to cross the street. When pushing the button, the light would immediately turn to red. Then it would resume its normal functioning. Can you redraw the diagram from above to represent this new situation? What would need to be added?

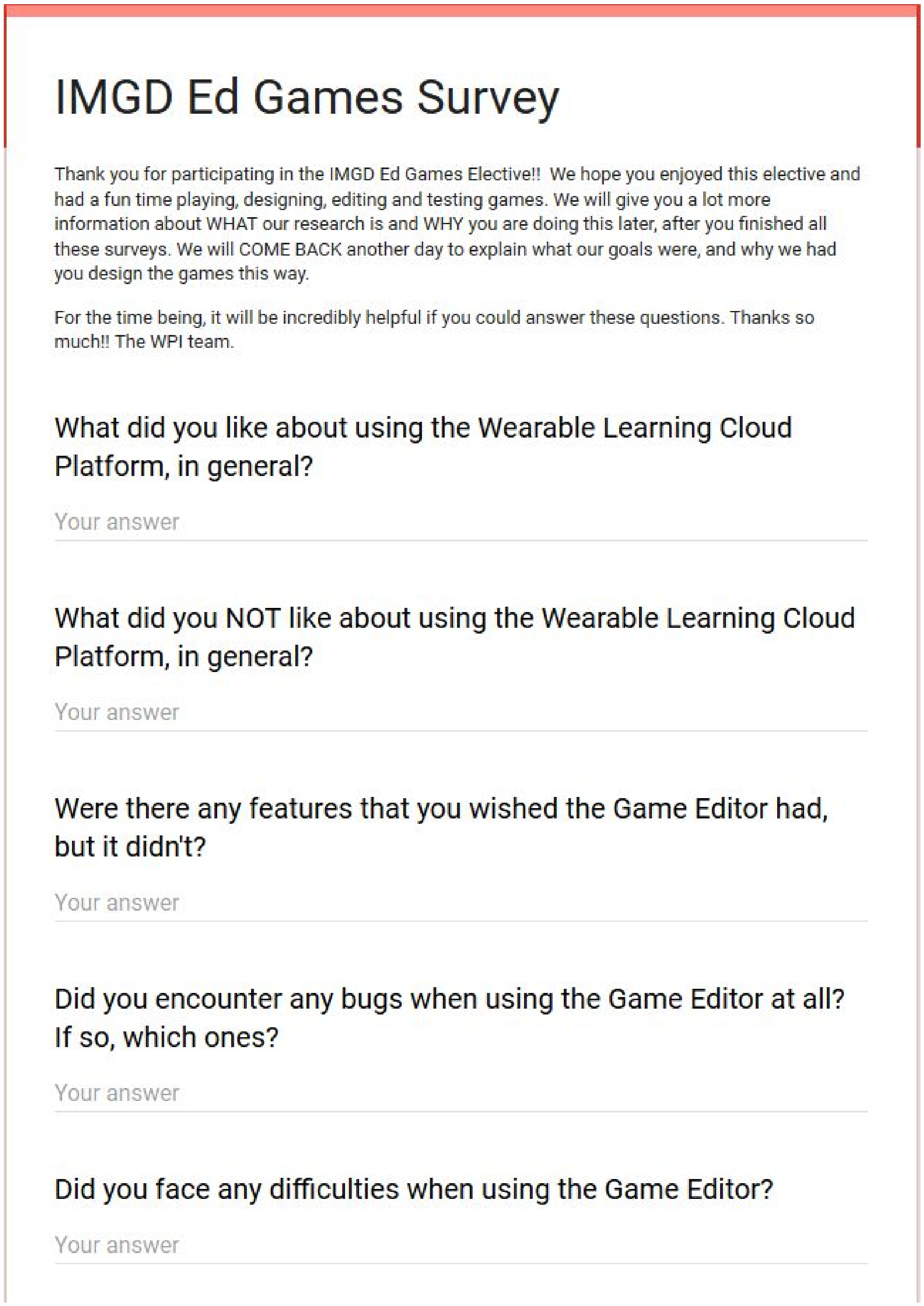
Could you design a diagram like the ones before to represent this new situation?

3.) Draw a diagram that will represent the output of the digital display of an elevator and how it changes depending on which of two buttons are pressed. Make sure the diagram addresses:

- When hitting the “up” button, the digital display will show “Top Floor”
- When hitting the “down” button, the digital display will show “Bottom Floor”
- The button can be pressed when already at the top or bottom floor.

Draw your diagram below.

APPENDIX B. Survey about Perceptions of the WLCP's "Game Editor"



IMGD Ed Games Survey

Thank you for participating in the IMGD Ed Games Elective!! We hope you enjoyed this elective and had a fun time playing, designing, editing and testing games. We will give you a lot more information about WHAT our research is and WHY you are doing this later, after you finished all these surveys. We will COME BACK another day to explain what our goals were, and why we had you design the games this way.

For the time being, it will be incredibly helpful if you could answer these questions. Thanks so much!! The WPI team.

What did you like about using the Wearable Learning Cloud Platform, in general?

Your answer

What did you NOT like about using the Wearable Learning Cloud Platform, in general?

Your answer

Were there any features that you wished the Game Editor had, but it didn't?

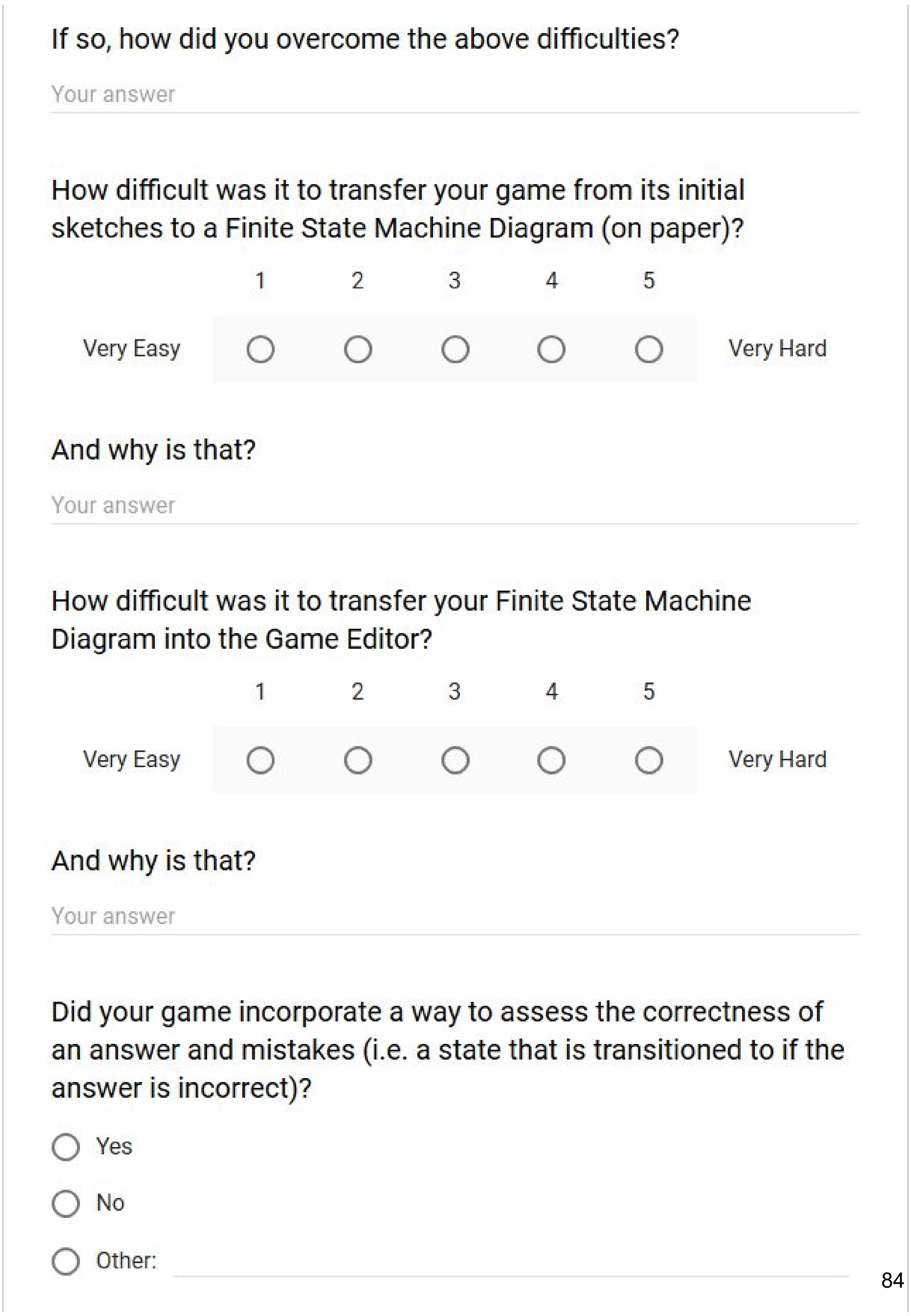
Your answer

Did you encounter any bugs when using the Game Editor at all? If so, which ones?

Your answer

Did you face any difficulties when using the Game Editor?

Your answer



If so, how did you overcome the above difficulties?

Your answer

How difficult was it to transfer your game from its initial sketches to a Finite State Machine Diagram (on paper)?

	1	2	3	4	5	
Very Easy	○	○	○	○	○	Very Hard

And why is that?

Your answer

How difficult was it to transfer your Finite State Machine Diagram into the Game Editor?

	1	2	3	4	5	
Very Easy	○	○	○	○	○	Very Hard

And why is that?

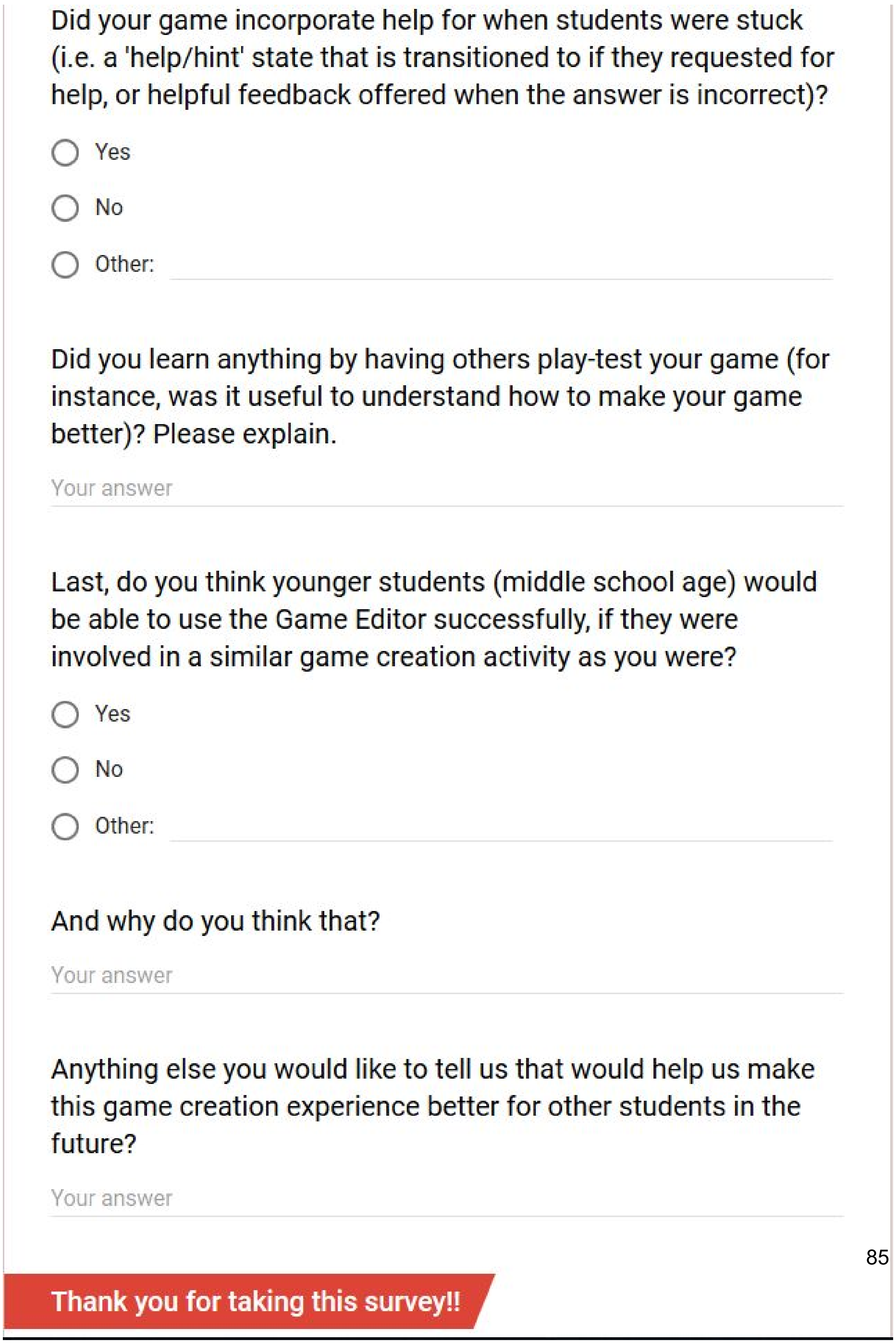
Your answer

Did your game incorporate a way to assess the correctness of an answer and mistakes (i.e. a state that is transitioned to if the answer is incorrect)?

○ Yes

○ No

○ Other:



Did your game incorporate help for when students were stuck (i.e. a 'help/hint' state that is transitioned to if they requested for help, or helpful feedback offered when the answer is incorrect)?

Yes

No

Other:

Did you learn anything by having others play-test your game (for instance, was it useful to understand how to make your game better)? Please explain.

Your answer

Last, do you think younger students (middle school age) would be able to use the Game Editor successfully, if they were involved in a similar game creation activity as you were?

Yes

No

Other:

And why do you think that?

Your answer

Anything else you would like to tell us that would help us make this game creation experience better for other students in the future?

Your answer

Thank you for taking this survey!!