

Dakota State University

Beadle Scholar

Masters Theses & Doctoral Dissertations

Fall 12-2019

Escape Puzzler

Shane Robertson

Follow this and additional works at: <https://scholar.dsu.edu/theses>



Part of the [Computer Sciences Commons](#)



ESCAPE PUZZLER

A graduate project submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Information Systems

December, 2019

By

Shane Robertson

Project Committee:

Dr. David Bishop

Dr. Christopher Olson

Dr. Ronghua Shan



PROJECT APPROVAL FORM

We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a project for the degree of Master of Science in Information Systems.

Student Name: Shane Robertson

Master's Project Title: Escape Puzzler

Faculty supervisor: Dave Bishop Date: 12/17/2019

Committee member: Chris Olson Date: 12/16/2019

Committee member: Rozhna Khan Date: 12/16/2019

ACKNOWLEDGMENT

First and foremost, I would like to thank my three children for providing inspiration and motivation to help fuel the nature of the project.

I would like to thank Daniel Thomas (<http://www.danielthomasart.com>) for providing the tile graphics used throughout this project.

I would like to thank Toke Game Art (<http://tokegameart.net>) for providing the graphics for the sprites.

I would like to thank Angelina Avgustova (<https://www.gamedevmarket.net/asset/loot-icons-8786/>) for providing additional graphics for the project. These include keys, lanterns, and various props.

I would like to thank Tao & Sound (<https://www.gamedevmarket.net/member/tao-sound/>) for providing audio assets for the project. This includes music and sound effects.

I would like to thank Dr. David Bishop for his participation as committee chairman for this project.

I would like to thank Dr. Christopher Olson for his participation on the project committee.

I would like to thank Dr. Ronghua Shan for his participation on the project committee.

ABSTRACT

This project is a systems design project. The goal of the project was to complete an online game for the purpose of entertaining an end-user. There is also additional research potential with analyzing the end-user behavior. This project showcases various skills learned at Dakota State University. This project required systems analysis, research of information technologies, database design, and project management.

About 50% of the planning phase was dedicated to scanning the IT industry for various technologies. We researched web hosting providers that would support the technologies we wanted to use. Also, a survey of available game engines was conducted. Additionally, all of the creative elements of the game were drafted during the design phase. Game mechanics were also contemplated. We decided to outsource the game assets, such as graphics and audio. However, no single vendor could provide everything needed. This led to a careful consideration of purchases to ensure that all of the game assets would weave together seamlessly. Finally, time was dedicated to create several technical artifacts, such as wireframe diagrams, entity relationship diagrams, and UML class diagrams.

During the implementation phase, all of the planning was put into action. There were three main components created: the client, the MVC web application, and the class library that contained all of the core game logic. A supplementary unit / integration project was also created dedicated to testing.

The end result was a finished game. Escape Puzzler is now live on <https://www.escapepuzzler.fun> and it is currently being tested. Once testing is complete, another iteration of development will help refine and polish the game even further.

DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

A handwritten signature in blue ink, appearing to read "Shane Robertson", is written over a horizontal line.

Shane Robertson

TABLE OF CONTENTS

PROJECT APPROVAL FORM	II
ACKNOWLEDGMENT	III
ABSTRACT	IV
DECLARATION	V
TABLE OF CONTENTS	VI
LIST OF TABLES.....	VII
LIST OF FIGURES.....	VIII
INTRODUCTION	1
LITERATURE REVIEW	2
SYSTEM DESIGN	4
GAME DESIGN	4
TECHNICAL DESIGN.....	9
PROJECT MANAGEMENT	33
PROJECT PHASES	33
PROJECT MANAGEMENT PROCESS.....	34
USER STORY LIST.....	35
CONCLUSIONS.....	42
OBSTACLES	43
REFERENCES	47
APPENDIX A: BROWSER COMPATIBILITY	49
APPENDIX B: USER MANUAL	50
APPENDIX C: LICENSE AGREEMENTS.....	53
TOKE GAME ART.....	53
STANDARD GAMEDEV MARKET LICENSE	54

LIST OF TABLES

Table 1. Reference Games	9
Table 2. Chosen Technologies	9
Table 3. User Story List	41
Table 4. Browser Compatibility	49
Table 5. Menu Options.....	52

LIST OF FIGURES

Figure 1. Class Diagram - Main Core Classes	15
Figure 2. Class Diagram - Managers	16
Figure 3. Class Diagram - Repositories	18
Figure 4. Class Diagram - Tiles	18
Figure 5. Class Diagram - Playable Objects 1	19
Figure 6. Class Diagram - Playable Objects 2	20
Figure 7. Class Diagram - Playable Objects 3	21
Figure 8. Class Diagram - Constants	22
Figure 9. Class Diagram - Action Script Core	23
Figure 10. Class Diagram - ActionScript Conditions	24
Figure 11. Class Diagram - Action Consequences.....	25
Figure 12. End User Menu Flowchart.....	31
Figure 13. Data Flow Diagram - Client and Server Communication	32
Figure 14. Negate y-coord to align multidimensional array to quadrant IV	45

CHAPTER 1

INTRODUCTION

Background of the Problem

This project is a systems development project. The system is an application of gamification. It uses entertainment and game design patterns to promote critical thinking. The game genre is a casual logical puzzle game with elements of an adventure game. The puzzles will present the end-user with a series of challenges for either critical thinking, pattern recognition, or memory.

Objectives of the project

The goal of the project is to complete an online game that entertains the end-user. As a minor complementary objective, telemetry is inserted into the game to analyze end-user difficulty. One reason for this is to understand end-user usage to make future improvements. Another reason is to potentially convert this project into a tool for further research correlating end-user behavior with cognitive abilities and overall personality traits.

CHAPTER 2

LITERATURE REVIEW

Before the project commenced, we contemplated several possibilities for a system design project. This brainstorming session led to the decision to write a video game. In one sense, writing a game would provide a good illustration of technical capabilities. It would require several skills that exemplify a comprehension indicative of what is learned from the master's program. It would require systems analysis, research of various new information technologies, RDBMS design, and project management. A game would require graphics rendering alongside client/server communication with score computations and database queries in the background all in real-time. Unlike other software creations, a game poses an additional challenge because the application domain did not seem to have any reliable template or standard operating procedure. With accounting systems, a programmer can rely on GAAP. ERP design might be able to lean on established operating procedures not yet digitized. With created a game, there is no template. However, in deeper research, game design patterns and methodologies were discovered. (Rogers, 2014) (Scolastici & Nolte, 2013)

Additionally, video games uniquely lure end-users in to become deeply engaged and immersed. Creating a game that provides an educational benefit to the end-user, for example, can be achieved. (National Research Council, 2011) (Edvardsen & Kulle, 2010) (Fisher, 2015)

A puzzle game had a manageable time frame within semester requirements and potentially had the capability to provide both of the original requirements (analysis capabilities and educational) without needing to specialize in either domain.

We knew that we wanted to program for the ASP.NetCore platform and host on Azure. This was inspired by the industry and resources from the INFS 730 course (Freeman, 2017).

We did not want to reinvent the wheel with game-related programming, so we decided to pick an established game engine. The requirements for the game engine were as follows: it must be stable and mature, it must be relatively popular, it could not be expensive, it had to work with the decided server technologies (Azure and ASP.Net Core), and it had to work within the realm of HTTP and HTML. We researched many different game engines, but eventually narrowed the research down to three different products: GameMaker, Godot, and Unity.

GameMaker seemed interesting, but it had several flaws. It did not have the reputation as a serious professional tool. Additionally, the programming language that it used, GML, was a proprietary language that would have no use outside of the platform. It was useful for making 2D games, but was missing key 3D features. When compiling to HTML, it relied on the Canvas tag and JavaScript, which did not seem as powerful as possible in today's market. These limits were also considered due to versatility for potential future projects. (Elliott, 2013)

Godot was another interesting find. It seemed to have more power than GameMaker and it allowed C# as a scripting language. It had the additional benefit of being free and open source. However, Godot is still fairly new, only being five years old. Unity, in contrast is fourteen years old. GameMaker is twenty years old. Godot lacks a mature following associated with many successful technologies. It has a lot of good news associated with it. However, that in itself, may not be a good thing. We suspect that such news indicates that Godot is still early in the Gartner Hype Cycle. We will continue to keep an eye on it to see how it develops and might consider it for other projects in the future.

Unity was ultimately chosen for development. Unity is the most successful out of the three choices. It is actually one of those most popular game engines in the market. It is more powerful than GameMaker (and probably Godot). It can target over a dozen different platforms. It works well with either 2D or 3D technology. It has associations with other large IT companies. The scripting language that it uses is C#, lowering barrier to entry and maintaining a smaller learning curve. There also exists a wealth of knowledge associated with it. (Unity Learn Premium, n.d.) (Cohen, 2016) (Jeffrey, 2017) (Lander, 2019) (Somasegar, 2015)

CHAPTER 3

SYSTEM DESIGN

This system design chapter is broken down into two sub-chapters: game design and technical design. The game design covers high-level design patterns and explores the creative aspects of the game. It also serves as tool for understanding the game from the perspectives of non-technical professionals, such as artists and marketers. A game design component is a standard in the game production industry. The technical design, in contrast, focuses on IT-related components and decisions related to the project.

Game Design

Game Design Introduction

This escape puzzler is a game where a player tries to use pattern matching skills, memorization, and his/her surroundings in a room to solve the room's puzzle. The goal of each room is to walk through the exit door successfully. There is no concept of combat or fighting in this game.

Genre

The genre of the game is a casual puzzle game with a few elements of an adventure game laced into it. It is meant to be picked up on a lunch break to provide some mental stimulation. A single room should not take more than ten minutes to complete. It is a single-player game. The style of graphics will be retro cartoonish. The end-user view is top-down 2D tile set.

Characters

The main character is Crona, an androgynous ninja. A secondary character is Lyric, the mapmaker of the levels and the main antagonist of the story.

Setting

The game is timeless and embeds anachronistic elements into one setting.

Graphics

The graphical interface is a 2-dimensional top-down tile set. All artwork is outsourced so that the project can focus on technical design and development. The style of the artwork will all conform to a look of a cartoonish atmosphere.

Tile Set

The tile set graphics are purchased from Daniel Thomas. (Thomas, n.d.) The tile set is a hand-painted set of drawings for elements of a dungeon. There are three variations in the color theme: purple, blue, and gray.

Sprites

While some additional graphics from Daniel Thomas are used for sprites, the graphics associated with the sprite of the main character is purchased from Toke Game Art (<http://tokegameart.net/>). Each sprite will animate when in movement and each direction will appear with a smooth transition.

Miscellaneous Graphics

Other miscellaneous graphics are purchased from other vendors from the GameDev.net asset store. (Avgustova, n.d.) These include keys, lanterns, and various props to sprinkle around the dungeon floors and bring the levels to life.

Gameplay Features

Scoring System

The system uses several variables to determine the score earned after completing a room:

- Gems collected: The more gems uncovered, the higher the score
- Lantern usage: The less a lantern was used, the higher the score
- Duration: Date/time entering the exit door minus date/time starting the level

Player Character Navigation

The player begins each room in a predefined location unique to each room. The main objective of each room is to transcend to the exit door successfully. The player uses the regular arrow keyboard keys (up, down, left, right) to control the player's character to navigate around. Obstacles, such as walls or braziers, usually block the playing character's progress.

Collecting Items

In order for a player to collect items, he/she must control the playing character to navigate to the item. Once the playing character stands over the item, the item will seemingly disappear from the room floor and appear in the character's inventory instead.

Keys

A key is an item that the player must sometimes discover and pick up before unlocking a locked door. It acts as an intermediate task to complete a level. One a key is used to open a door, that door is forever open and the key disappears from the player's inventory. A key is only useful for the level in which it is found. Obtaining a key and using a key will both trigger server-side events to help track player accomplishments and their skill.

Lanterns

For levels shrouded in darkness, a lantern acts as a method for removing the obstacle of the darkness. Light from a lantern is temporary – it goes out after a designated time. However, the player can re-light the lantern. Not only will a lantern acquisition be used as a milestone in a level, but the usage will help determine the player's score. This is based on the player's memory. Each time a lantern is used, a server-side event is recorded. The more frequent a player uses a lantern, the less the player's score. A lantern can also be used to illuminate secrets, such as hidden text or hidden passageways.

Braziers

A brazier is completely immobile and impassable, so the player is not able to walk through it. However, in levels with low visibility, a brazier produces a little bit of heat. This allows a player to progress through a level. Reaching a brazier can act as milestones as the player progresses through a level.

Gems

Gems are extra items found in rooms. They are not mandatory to collect, but they do provide extra points if the player successfully exits a room. As such, they add a bit of replay value to the rooms when a player attempts to return to previously explored rooms. They are often hidden and require special attention for a player to go out of their way to uncover.

Exit Doors

An open exit door is the main goal of each level. Passing through an exit door marks the completion of the room. However, many exit doors will be locked and will require a key,

located somewhere else within the level. When a player walks through an exit door, a server-side event will be recorded and the system will record the player's progress.

Treasure Chests

Treasure chests are another intermediate level achievement. Treasure chests are often initially locked and must be unlocked by some mechanism in the room. Their contents vary by room, but they usually hold a gem or a key.

Floor Buttons

Floor buttons are another intermediate level achievement. They act as mechanisms for opening or unlocking other items, such as treasure chests or locked exit doors. Usually when a floor button is present in a room a player must first navigate to the floor button to press it by walking over it in order to unlock the next step in a room. Some floor buttons stay pressed when initially stepped on. However, other floor buttons will re-lock once depressed. In this situation, a player must push a block on the floor button in order to keep it from re-locking.

Blocks

The basic obstacle of the game is a simple block. Blocks have several variants to them. Some blocks are completely immobile and prevent the player's movement. A player will be unable to walk through the block. Other blocks can be pushed forward indefinitely until they press up against something else immobile, such as a wall or another block. Other blocks can only be moved for a finite number of tiles until they become completely immobile.

In order to push a movable block, a player needs to walk up to a block and continue walking forward. If a block is capable of being moved, the block will move forward along with the player's movement. While pushing blocks forward, a player's movement is hindered and the player moves more slowly. Blocks can only be moved forward, not pulled backwards. Thus, careful consideration must be taken regarding the direction a player decides to push a block. Some rooms are designed where block movement order is also important.

In some cases, a block is not merely an obstacle, but a foundational tool for pattern matching or solving problems. For example, there is a level where the exit door only opens if a floor button is pressed. The player can step on the button to open the door. However, as soon as the player walks off of the button the door closes again. The solution is to push a block over the button.

In another example, certain blocks might need to be pushed in a specific pattern in order to unlock the exit door.

Illusion Walls

On rare occasions, certain walls will be imaginary. If a player attempts to walk through a wall then he/she will be able to do so without any difficulty. This might be the only way to progress through a level. Secret passages through walls act as obstacles by exploiting a player's assumptions in their capabilities. There might be some visual hints that a wall is an illusion, such as a carpet leading to nowhere or the illusion wall may be surrounded by braziers. Illusion walls are a test of the player's ability to comprehend surrounding cues and experiment with his/her surroundings.

Darkness

When the player is unable to see, progress through a room becomes much more difficult. A player might stumble around, hitting blocks and walls without ever knowing it. Darkness can be diminished with braziers or with lanterns.

Time

While not usually an obstacle, room progression will be timed and used as a basis for scoring. Some players might want to use time as a metric of competition against themselves to beat a level faster than before. The less time a player needs to successfully complete a room, the higher the player's score when room completion is achieved.

Locked Doors

Many exit doors will be locked and will require a key to unlock them. Opening a locked door is essentially the last obstacle in a room. Once a door is unlocked a player can walk through the door and win the level.

Target Audience

The target audience is a casual gamer interested in playing something simple but thought-provoking within small quantities of time.

Reference Games

The following games are similar in tone and style to Escape Puzzler. These games are listed as a reference to provide an understanding of what to expect and the surrounding market:

<i>Product Name</i>	<i>Platform</i>	<i>Released</i>
Bombberman	Nintendo Entertainment System	1985
Adventures of Lolo	Nintendo Entertainment System	1989
A Good Snowman is Hard to Build	iOS, Android, Windows, Linux, OS X	2015
Maze of Adventures	Windows	2018

Table 1. Reference Games

Technical Design

The following table summarizes all of the relevant technical decisions for the project:

<i>Category</i>	<i>Selection</i>
Server-side programming language	C#
Server-side platform / model	ASP.Net Core MVC
Client-side platform	HTML, JavaScript, Web Assembly, WebGL
Client-side programming	HTML, JavaScript, C#
Game development engine	Unity
Web hosting	Microsoft Azure
Domain registration / DNS	NameCheap.com
Chosen domain name	www.escapepuzzler.fun

Table 2. Chosen Technologies and Services

Server details

The server-side code is written in C#. It is separated into two different projects, a class library for the core logic and domain code and an MVC web application to handle the web server requests.

The domain name registered was www.escapepuzzler.fun. The domain registrar used is named NameCheap.com. That domain name registrar was chosen because they offered free WHOIS masking and cheap domain name registration. NameCheap is also providing Escape Puzzler's name server services. We had to manually adjust the CNAME and A DNS records to point to Azure's web hosting servers.

MVC

The Model-View-Controller (MVC) web app is essentially the glue that connects the game logic on the server to the client. It handles the incoming HTTP requests from the client.

The MVC controller passes the end-user's commands to the game logic and receives back any changes. It then sends those changes back to the end-user. It is also responsible for invoking the repository object and calling its save method to save the game state for later. The following are the main methods of the controller applicable to Escape Puzzler:

```
[Authorize]
public IActionResult Index()
{
    if (Models.Admin.SystemControl.IsOffline)
        return View("Offline");

    return View();
}
```

This index method calls the view of the main page. The view includes the web assembly to load the actual game. Alternatively, if the system has been switched offline by an administrator then it returns the “Offline” view, which provides a user-friendly message to ask the end-user to try again in a few minutes.

```
public IActionResult GetRoomList()
{
    var roomList = roomRepo.GetRoomSummaries();
    return Ok(roomList);
}
```

When the client enters the “Get Room List” scene, the client makes a call to this method to get a summary of the rooms. It produces a room summary result, which is a data transfer object that contains the room name and unique ID. This array is passed to the client, which it uses to dynamically create a list of buttons.

```
public IActionResult GetRoom(int Id)
{
    var userId = GetIdentity();
    var room = roomRepo.GetRoom(Id);
    var gs = new EscapePuzzler.Core.Entities.GameState(room, userId);
    gsRepo.SaveGameState(gs);
    var returnVal = ClientRoomManager.CreateClientRoom(gs);
    return Ok(returnVal);
}
```

The `GetRoom` method is called when an end-user initially selects a new room to play. This method gets all of the data for the room and uses the data to create a game state. The game state is immediately saved. The game state is then converted in to a `ClientRoom` object (for purposes of a DTO) and the `ClientRoom` object is sent to the client. Once the client receives the `ClientRoom` object, it renders the room to the screen.

```
[HttpPut]
public IActionResult SendCommand(string GameStateId, string Command)
{
    var userId = GetIdentity();

    var gs = gsRepo.GetGameState(int.Parse(GameStateId));
    if (gs.Status == GameStateStatus.InProgress)
    {
        gs = GameStateManager.ProcessCommand(gs, Command);
        gsRepo.SaveGameState(gs);
    }
    var clientResult = ClientRoomManager.CreateClientRoom(gs);
    clientResult.broadcastMessage=Models.Admin.SystemControl.BroadcastMessage;
    return Ok(clientResult);
}
```

The `SendCommand` method is where most of the transactions take place. This is the method that is called when the end-user makes commands. The commands get sent to the `SendCommand` method. The game state gets retrieved from the repository. The `SendCommand` method then calls the `GameStateManager.ProcessCommand` method to process the command. The result, if anything changed, is re-saved using the repository once again. It then converts the game state into a `ClientRoom` object and sends the object back to the client. The client, at this point, re-renders the screen based on the updated `ClientRoom` object.

Authentication and Security

The server utilizes the stock identity management and authentication methods from the standard Microsoft ASP.Net Core library. It also leverages Facebook authentication. It implements both of these through the .Net Core's middleware library. The Facebook set up first had to occur on Facebook, itself. After signing up for the Facebook for Apps service, Facebook provided a unique application ID and application secret code. These were fed into the Facebook authentication middleware to authenticate the integrity of Escape Puzzler when communicating with Facebook's OpenID identity provider.

When in production, Escape Puzzler uses HTTP Strict Transport Security. (Hodges, 2012) This means that HTTPS is forced upon the end-users, even when they explicitly attempt to use the clear-text HTTP. Azure provides the TLS certificate for the Escape Puzzler website through the DigiCert GeoTrust RSA certificate authority.

Core Game Logic

Most of the game logic takes place in a separate class library. The MVC app relies on it to make decisions within its domain. However, the class library, itself, is independent and could potentially be placed anywhere in order to duplicate the game logic necessary to complete its task. For example, the presentation layer could be replaced by a Xamarin application and the core game class library could be placed along side it. This class library is composed of several different types of classes. These include entity classes, manager classes, repository classes, and helper classes.

Entity Classes

Entity classes provide structures for the plain old CLR objects. These classes are used for pure data storage. They do not interact with the other components of the code, themselves. Instead they rely on repository and manager classes to do the heavy lifting. The main entity classes used by the core logic include the Room class, the GameState class and the ClientRoom class.

Room: The Room class stores all tiles, playable objects, and extra objects of a room. It also holds information regarding the beginning and ending of the room. It essentially acts as the initial game state when a player begins to play a room.

GameState: The GameState class is the heart of the game. Once a player picks a room and the playable component of the game starts, a new GameState instance is created based on the Room object. It is the class that stores all of the various unique elements of the specific instantiated gameplay. This includes the player's current position, all of the positions of all of the playable objects, any objects still hiding in the room, the current lighting of the room, the player's inventory (for example, whether or not he/she has obtained a lantern), and the various metrics associated with the score.

ClientRoom: The ClientRoom class is basically a data transfer object that represents all of the information from the GameState that needs to be conveyed to the client. It is trimmed down and excludes any secret information that should not potentially leak to the

player. One example of information that should not be conveyed to the client is the list of hidden and invisible objects. The ClientRoom class exists on both the server side and the client side. It is serialized and deserialized as a JSON object during the AJAX calls between the server and the client.

There exist many other entity classes beyond the main classes. Many of the remaining entity classes adhere to one (or more) of the following contracts: IRoomTile, IRoomPlayable, IActionCondition, and IActionConsequence.

IRoomTile: An IRoomTile object has an image number that corresponds to a tilemap element on the client side. It also has a Boolean variable to determine if it can be walked over or if someone would collide against it. All IRoomTile objects are stored in the GameState TileLayer property as a multi-dimensional array of IRoomTile objects. Examples of IRoomTile objects include floor objects and walls.

IRoomPlayable: An IRoomPlayable object is an object that can potentially interact with the player. It might render a graphic on the client or it might be completely invisible to the player. Because an IRoomPlayable object might change its location in the room, and potentially occupy the same location as other IRoomPlayable objects, each object includes independent properties for X and Y coordinates. It also has a mandatory method ExecuteMovementTrigger that will trigger when they player moves over the same coordinates as the IRoomPlayable object. IRoomPlayable objects might either be stored in the game state's PlayableLayer property or the HiddenLayer property as a list of IRoomPlayable objects. Examples of IRoomPlayable objects are blocks, braziers, chests, easy buttons, exit doors, gems, keys, lanterns, and tricky buttons.

IActionCondition & IActionConsequence: An IActionCondition object is an object that must be triggered for a consequence to occur. The IActionConsequence object takes the appropriate action as a result of the IActionCondition objects being triggered. The IActionCondition object is the first necessary ingredient for an action script. The IActionConsequence object is the second ingredient for an action script. An IActionConsequence can potentially require multiple IActionCondition objects to be triggered. The signature method for an IActionCondition object is IsActionConditionTriggered. The methods required for an IActionConsequence contract are ExecuteActionConsequence and UndoActionConsequence. Examples of IActionCondition

objects are easy buttons, tricky buttons, invisible buttons, and keys. Examples of IActionConsequence objects are treasure chests and locked exit doors.

Managers and Repositories

Manager, factory, and repository classes are responsible for saving, retrieving, forming, and manipulating entity objects. Repositories maintain all of the necessary data access layer functionality and act as gateways. The managers act as interactors and map all of the end-user interactions with the in-game entities. The managers are also responsible for reshaping the entities based on end-user input. They contain all of the necessary domain-specific game logic. Managers rely on and manipulate entity data from the repository classes in order to make decisions. Most game-specific user stories and unit tests manifest around the manager classes.

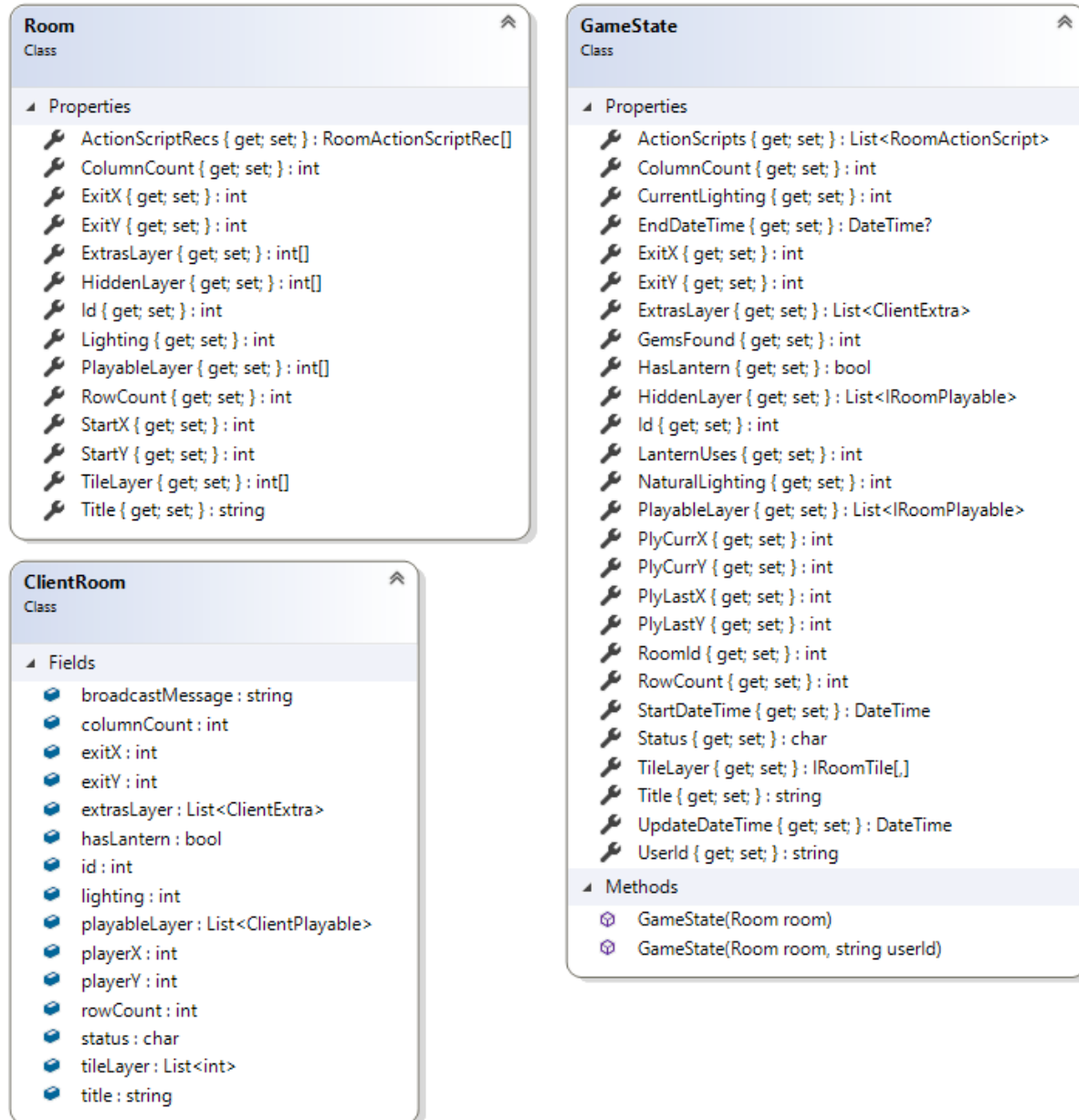


Figure 1. Class Diagram - Main Core Classes

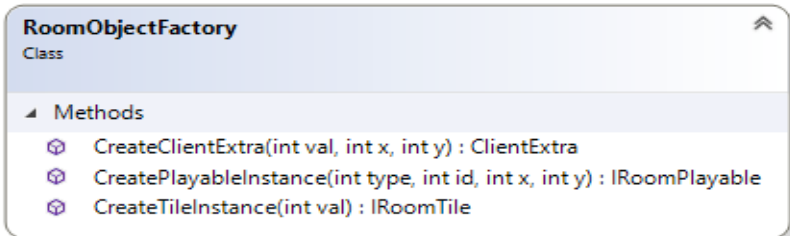
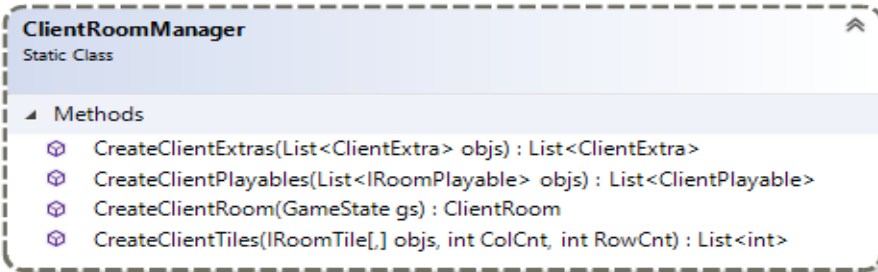
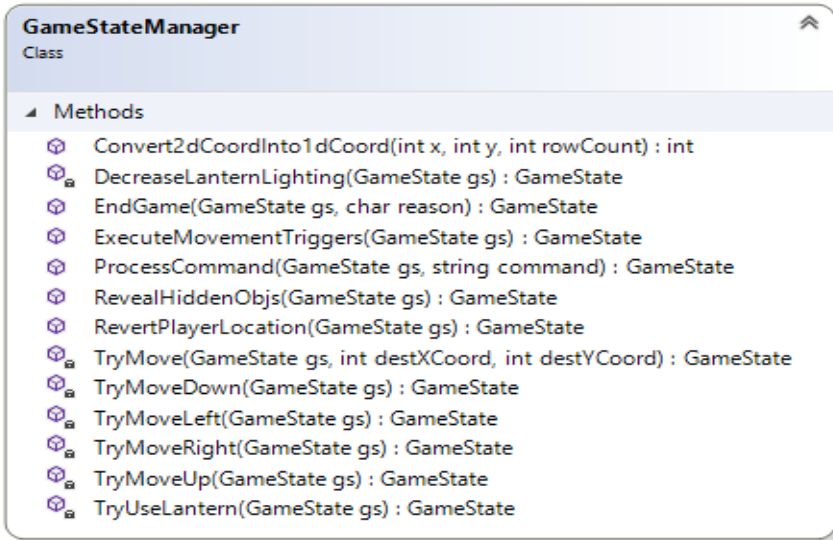


Figure 2. Class Diagram - Managers

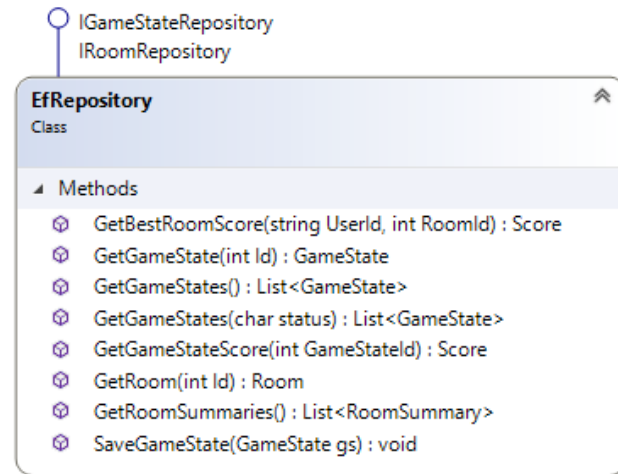
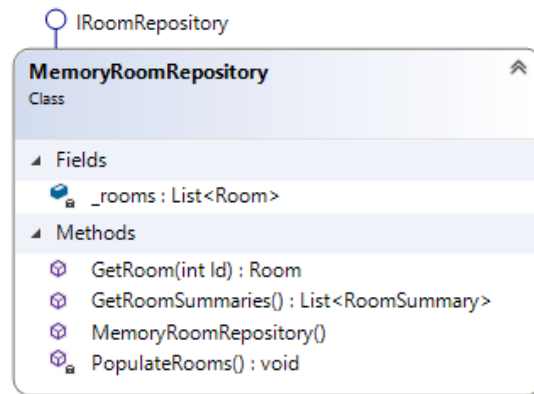
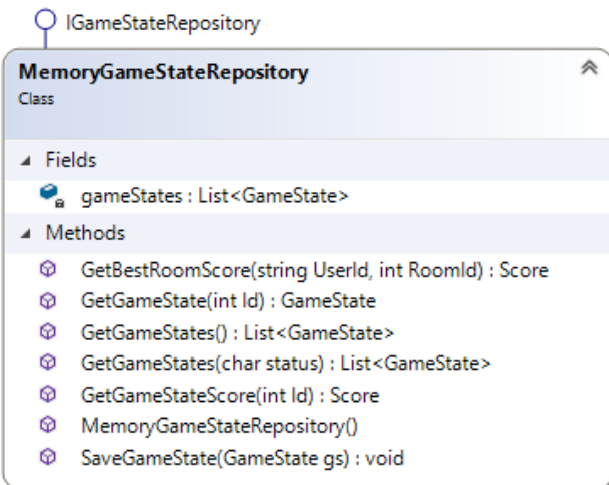
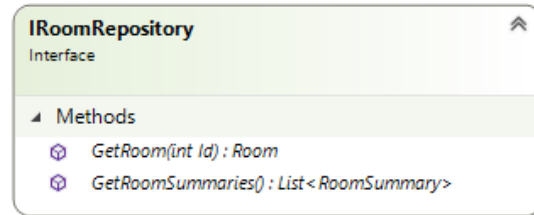
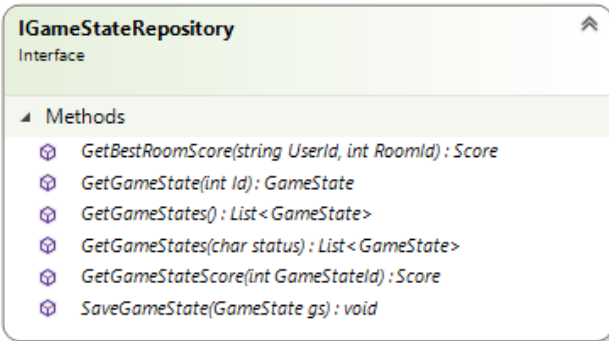


Figure 3. Class Diagram - Repositories

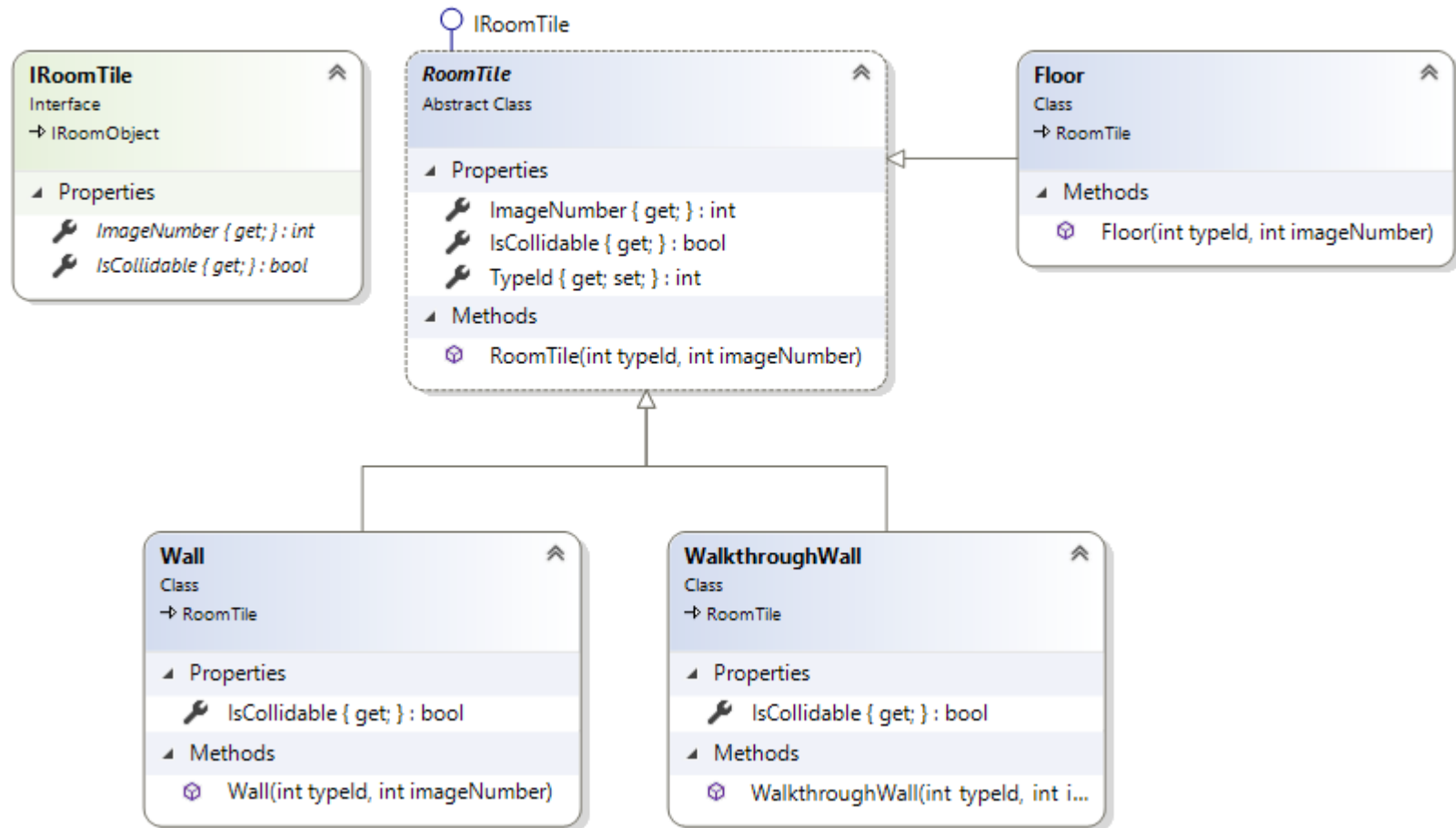


Figure 4. Class Diagram - Tiles

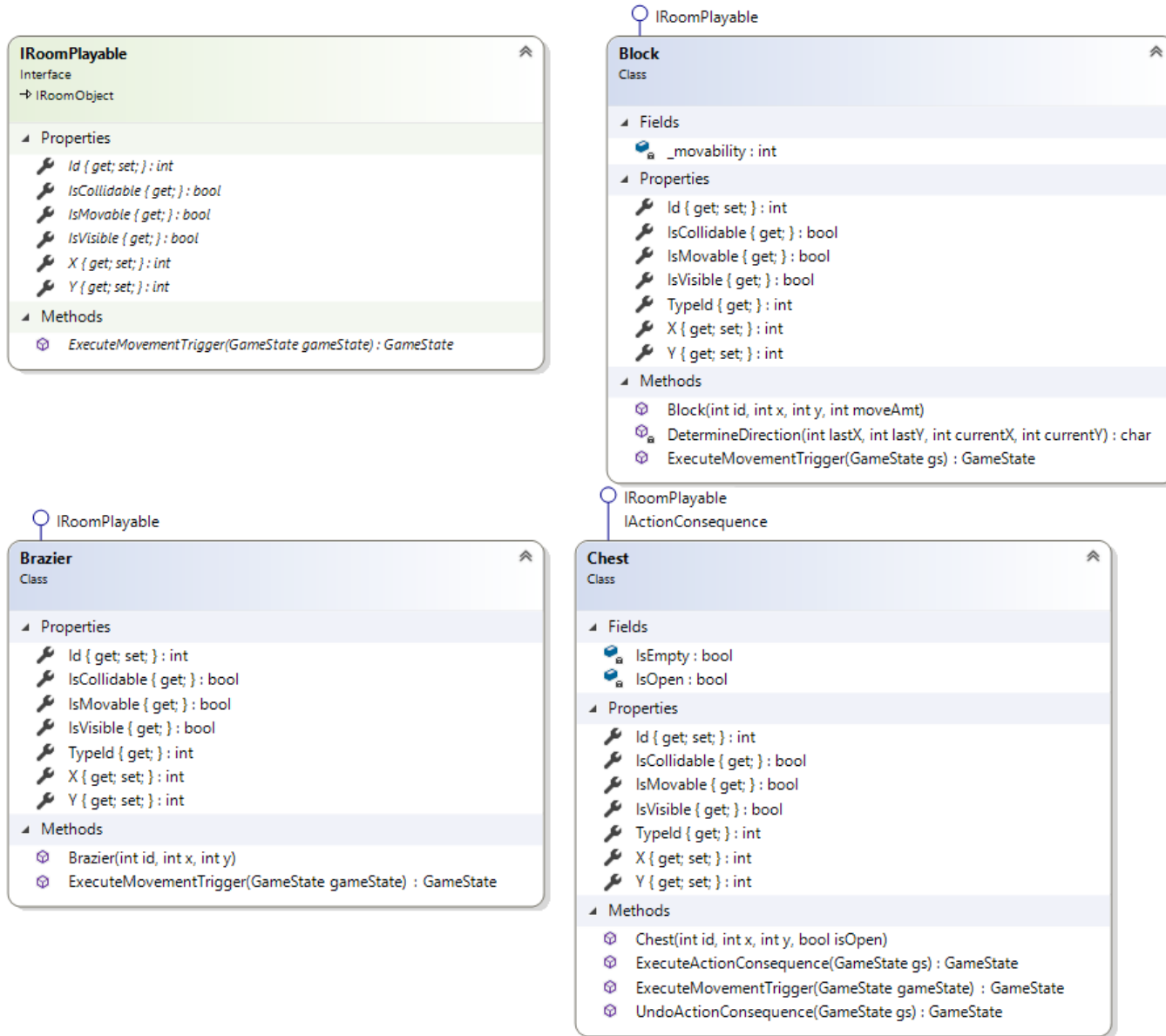


Figure 5. Class Diagram - Playable Objects 1

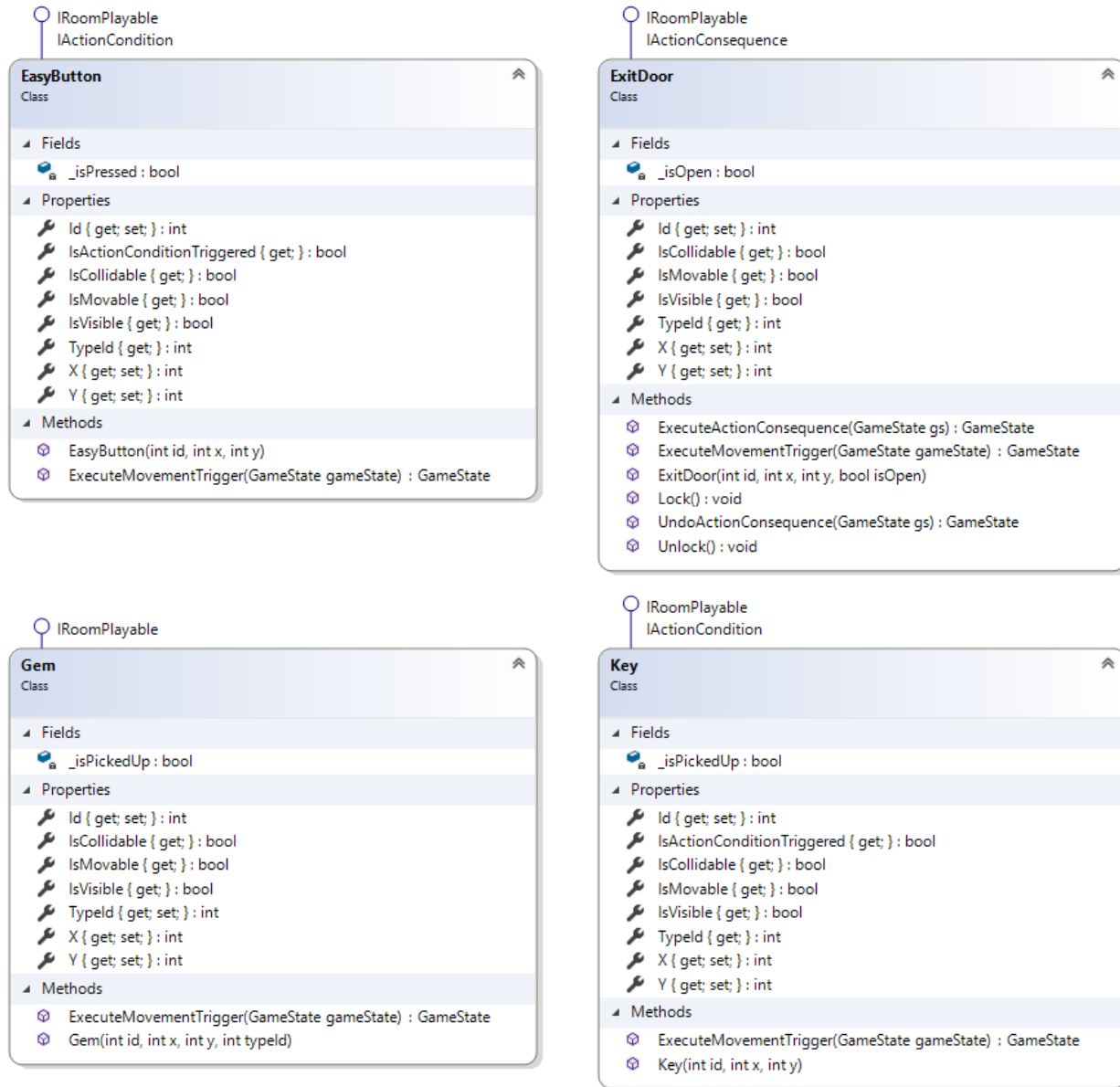


Figure 6. Class Diagram - Playable Objects 2

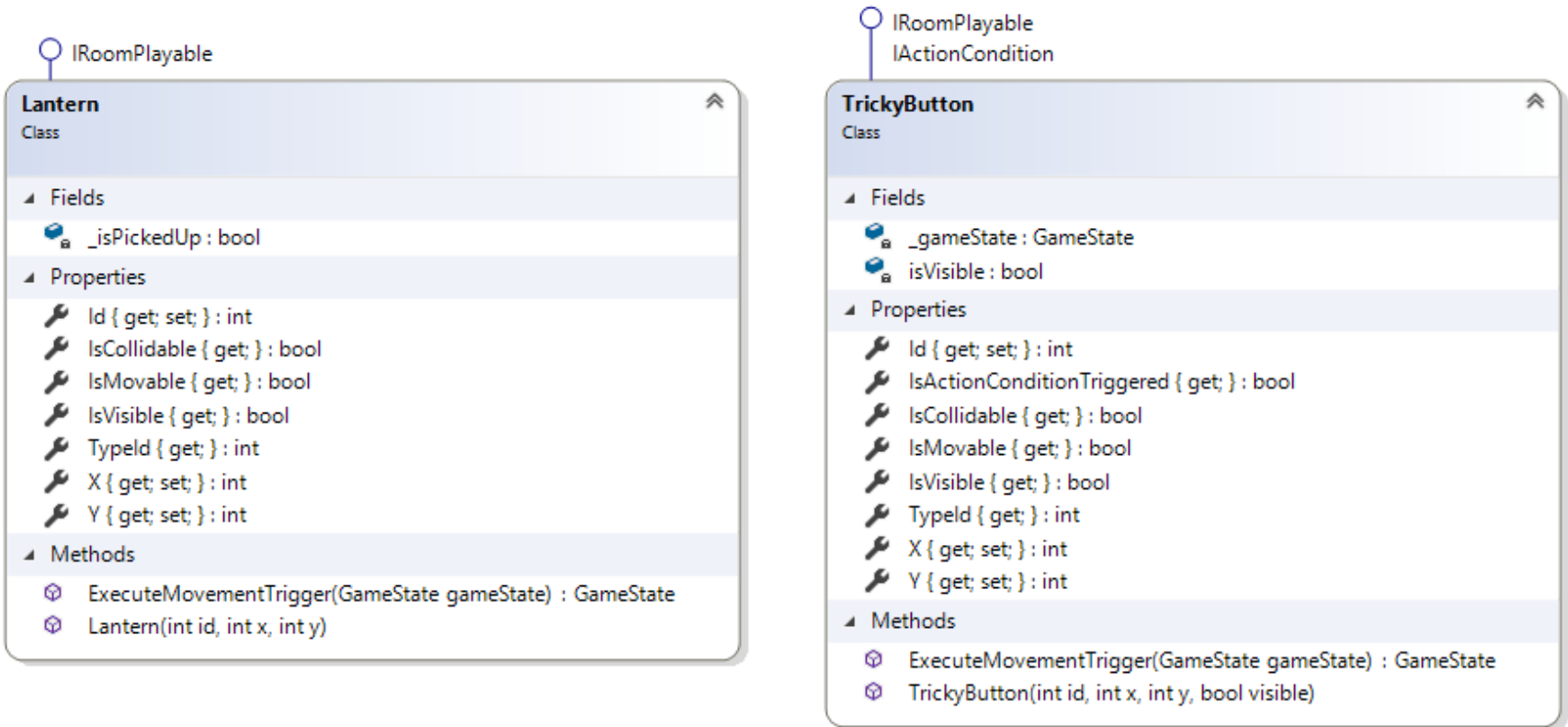


Figure 7. Class Diagram - Playable Objects 3

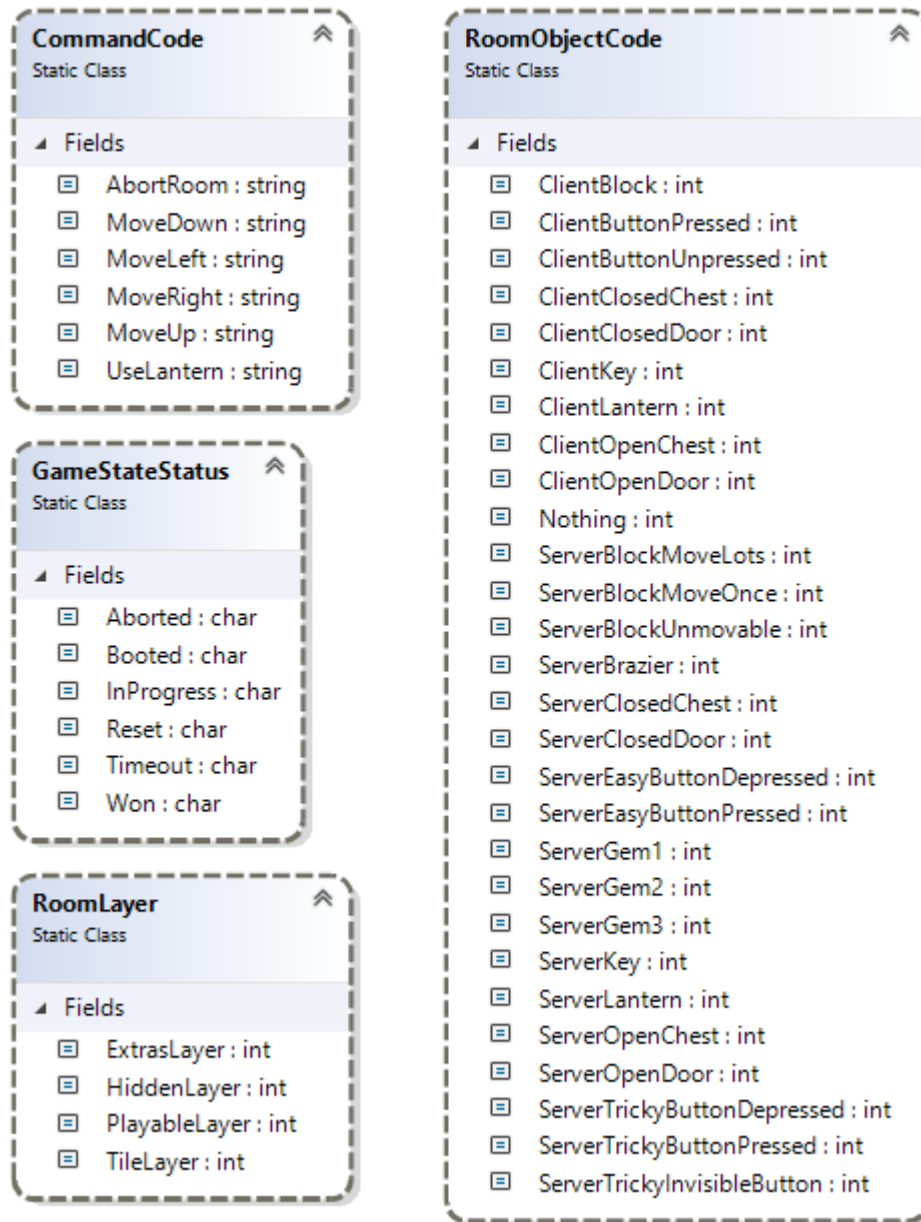


Figure 8. Class Diagram - Constants

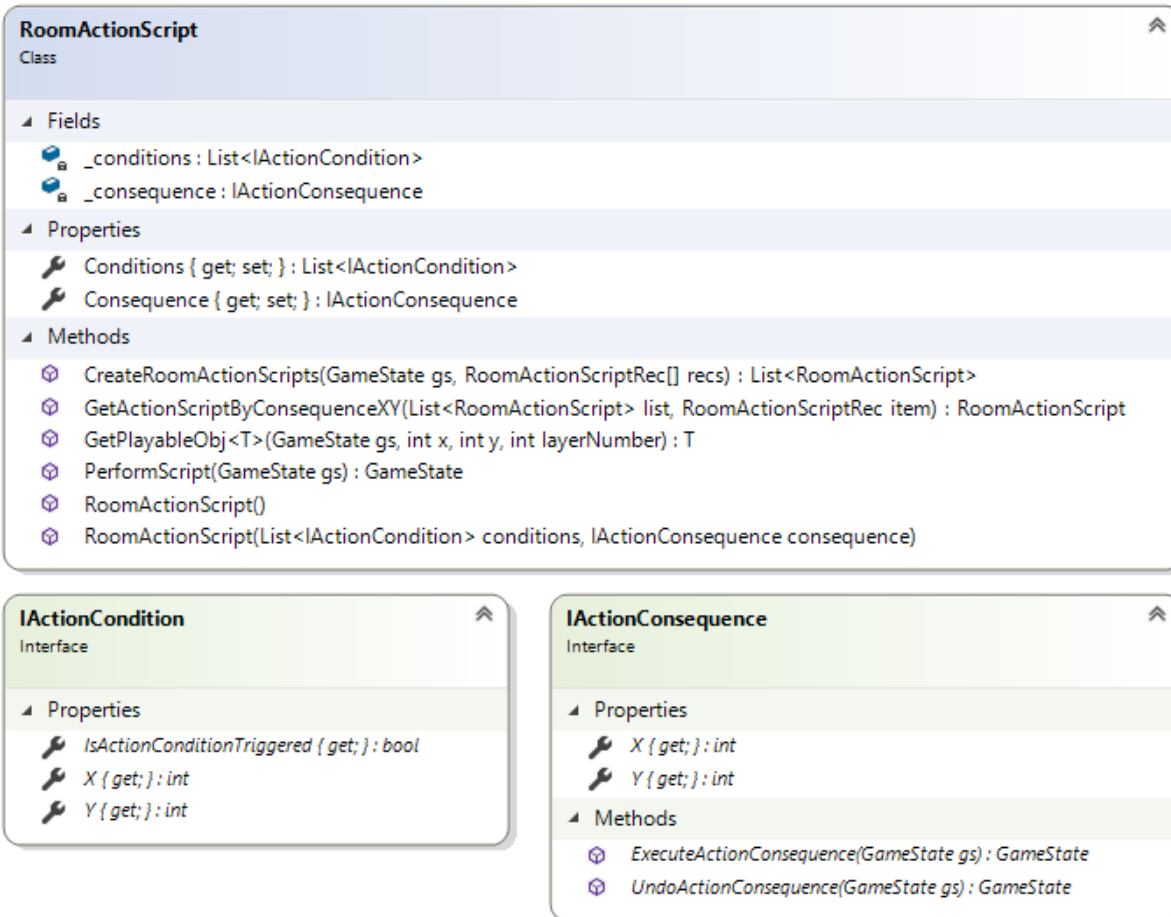


Figure 9. Class Diagram - Action Script Core

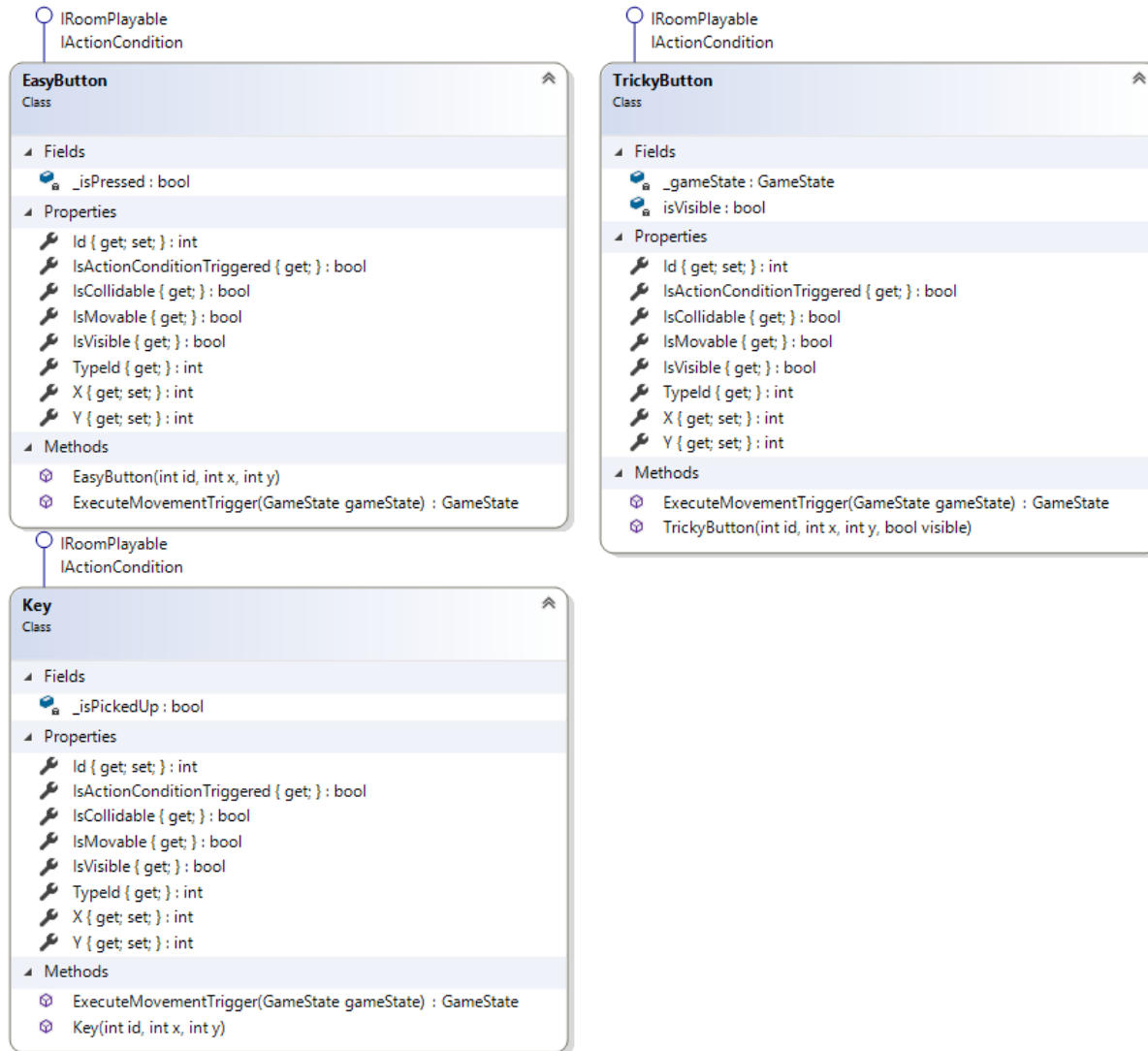


Figure 10. Class Diagram - ActionScript Conditions

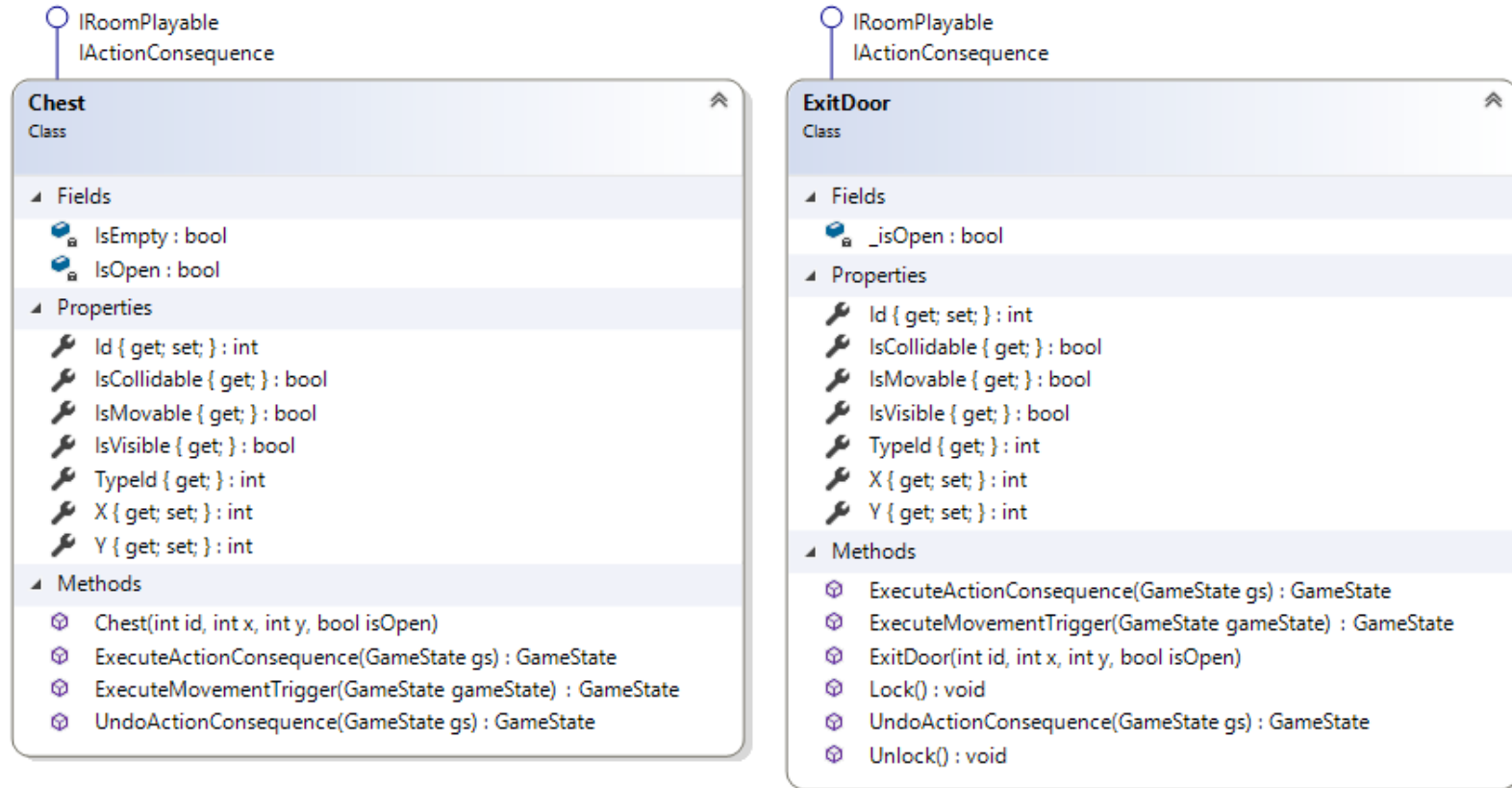


Figure 11. Class Diagram - Action Consequences

Client details

The client-side code is written in HTML 5, JavaScript, and Web Assembly. It uses a little bit of Bootstrap for CSS. However, hand-written client-side of the web site is minimal. The HTML and JavaScript is merely a framework for delivering the game. Most of the project focuses on the game. The game, itself, was created using the Unity Game Engine. By leveraging Unity, the end-result also includes Web Assembly and Web GL.

Choosing Unity as a game engine

It was decided in the beginning of the project that development would implement some game engine to assist with creating the game. Leveraging a game engine was an obvious choice because there was no need to reinvent the wheel. However, each game engine has its own unique limits and opportunity costs. Unity was ultimately selected. This choice was due to the following features:

- **Scripting language:** Under the hood, Unity uses C# for its scripting language. This provided a natural benefit because C# was the chosen language for the server-side of the project. Common technologies between the client and the server meant higher cohesion and a lower learning curve.
- **Popularity:** A technology is only as useful as the community that keeps it alive. Unity is one of the most popular game engines in the world.
- **Cost:** Unity is free to use.
- **Asset store:** The Unity platform has an immense library of gaming assets. The asset store includes graphics, audio, and additional mechanics. While other game engines have their own asset stores (such as Yoyo Games' Game Maker Marketplace) and third-party asset stores exist (such as the Game Dev Market) the Unity asset store was impressive enough to persuade us to evaluate the engine.
- **Prominence with Microsoft:** Microsoft indicates that they embrace Unity as a first-rate product for game development. (Lander, 2019) (Jeffrey, 2017)
- **Multi-platform:** Unity is able to easily compile to HTML5 / WebGL, iOS, Android, Windows, or Macs. Even though the initial target platform is only

the web, being capable of targeting multiple platforms without worrying about rewriting significant amounts of code is definitely desirable.

The Unity project is divided into four different scenes:

- **The intro scene:** This is the splash screen to help introduce the player to the game.
- **The room list scene:** This is the screen where the player gets to pick which room to play.
- **The dynamic room scene:** This is where most of the action takes place. The dynamic room queries the server for the room details and renders the server result. The player interacts by moving around and clicking on buttons. The dynamic room continues until either the player quits, returns to the room list, or wins the room.
- **The win scene:** This scene provides a summary of the player's score after the player wins a room.

The different layers and tile maps:

In Unity, all of the tiles and game objects are rendered on to a specific tile map and sorting layer. Escape Puzzler uses the following layers:

1. **Default:** This is the default location for all of the floor, carpet, and wall tiles.
2. **TilemapExtras:** This layer corresponds to extra graphics that should be rendered on top of the default layer. This includes decorative items on walls; like swords, shields, or flags; and colorful mood enhancers; such as blood or skulls on the floor.
3. **PlayableObjects:** This is the layer where all of the playable items render. The content is dynamically created as the ClientRoom object gets updated. Examples of objects on the PlayableObjects layer include blocks, braziers, buttons, chests, gems, keys and lanterns.
4. **Player:** This is the layer that the player character renders.
5. **AbovePlayer:** This is for objects that need to be rendered above the player. Currently, the only item in the AbovePlayer layer is the exit door archway.

Lighting

One of the elements in Escape Puzzler is dynamic lighting unique to each room. The natural lighting is stored on the server. This includes the Room object (in the integer Lighting property) and the GameState object (in the integer NaturalLighting and CurrentLighting properties). When the end-user attempts to use a lantern, the “Use Lantern” command is sent to the server. The GameStateManager processes the command. The end result, if successful, is that the room lights up. The actual rendering is done on the client-side. The ClientRoom object gets an updated lighting property value. The client, upon detecting this, makes a call to the following method:

```
public void SetAmbientLighting(int color)
{
    var tempAmbientLight = RenderSettings.ambientLight;
    float tempColorPercentage = (float)color / 255f;
    tempAmbientLight.r = tempColorPercentage;
    tempAmbientLight.g = tempColorPercentage;
    tempAmbientLight.b = tempColorPercentage;
    RenderSettings.ambientLight = tempAmbientLight;
    RenderSettings.ambientIntensity = 0;
}
```

The way that the client controls lighting is with the RenderSettings.ambientLight property. This is a built-in property that can change the background ambient light. It could also be useful for setting a certain mood by distorting a color in the entire scene. In this particular case, the server uses the light integer to set the RGB values. However, under-the-hood, the Unity engine expects the RGB values to be percentages instead of integers. With that in mind, the client method just converts it from its raw value from the server (0 – 255) into a percentage. The client method then applies the same value to the red, green, and blue values. This lightens (or darkens) the entire scene with an even color that does not distort any of the playable objects or tiles.

Playable Objects

Playable objects are treated differently than tilesets. Tile sets are static. They only need to be rendered once to the screen. They do not even need to be transmitted multiple

times because the client already knows of their existence and their placement from the initial game state load. Playable objects, in contrast, need to be consistently tracked and maintained because they can change. Treasure chests might open or disappear when accessed. Exit doors may or may not be locked. Items might be hidden at first but later revealed as a block gets pushed. All playable objects need to be checked every move. Because of this, each playable object has a unique identifier. After each command, the ClientRoom object is scanned for objects. Any object that exists on the client side that is not in the ClientRoom object is destroyed. Any object that is in the ClientRoom object, but not rendered to the screen, is instantiated. The following methods are used to control these tasks:

```

public void RemoveAbsentPlyObjInstances(List<ClientPlayable> plyObjs)
{
    var removeList = new List<int>();
    foreach(var thisObj in roomObjList)
    {
        if (!plyObjs.Exists(r => r.id == thisObj))
        {
            removeList.Add(thisObj);
        }
    }
    foreach(var thisObj in removeList)
    {
        roomObjList.Remove(thisObj);
        GameObject.Destroy(GameObject.Find(thisObj.ToString()));
    }
}

public GameObject CreatePlyObjInstance(int type, int id, int x, int y)
{
    GameObject o;
    var pos = new Vector3(x + 0.5f, (-1 * y) + 0.5f, 0);
    switch (type)
    {
        case 75:
            o = GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Gem1"),
pos, Quaternion.identity); break;
        case 76:
            o = GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Gem2"),
pos, Quaternion.identity); break;
        case 77:
            o = GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Gem3"),
pos, Quaternion.identity); break;
        case 80:
            // Lighting must be a bit distanced from everything else to take
effect.
            pos.z = -1.5f;
            o =
GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Brazier"), pos,
Quaternion.identity); break;
        case 81:

```

```

        o =
GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Lantern"), pos,
Quaternion.identity); break;
        case 91:
            o =
GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Block"), pos,
Quaternion.identity); break;
        case 144:
        case 192:
            {
                o =
GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/ExitDoor"), pos,
Quaternion.identity);
                var t = o.GetComponent<ExitDoorMgr>();
                t.Type = type;
                break;
            }
        case 300:
            o = GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Key"),
pos, Quaternion.identity); break;
        case 238:
        case 244:
            {
                o =
GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Chest"), pos,
Quaternion.identity);
                var c = o.GetComponent<ChestMgr>();
                c.UpdateImage(type);
                break;
            }
        case 245:
        case 246:
            {
                o =
GameObject.Instantiate(Resources.Load<GameObject>("Prefabs/Button"), pos,
Quaternion.identity);
                var c = o.GetComponent<ClientPlayableObj>();
                c.UpdateImage(type);
                break;
            }

        default:
            o = new GameObject(); break;
    }
    o.name = id.ToString();
    roomObjList.Add(id);
    return o;
}

```

The client end-user menu flowchart is illustrated below:

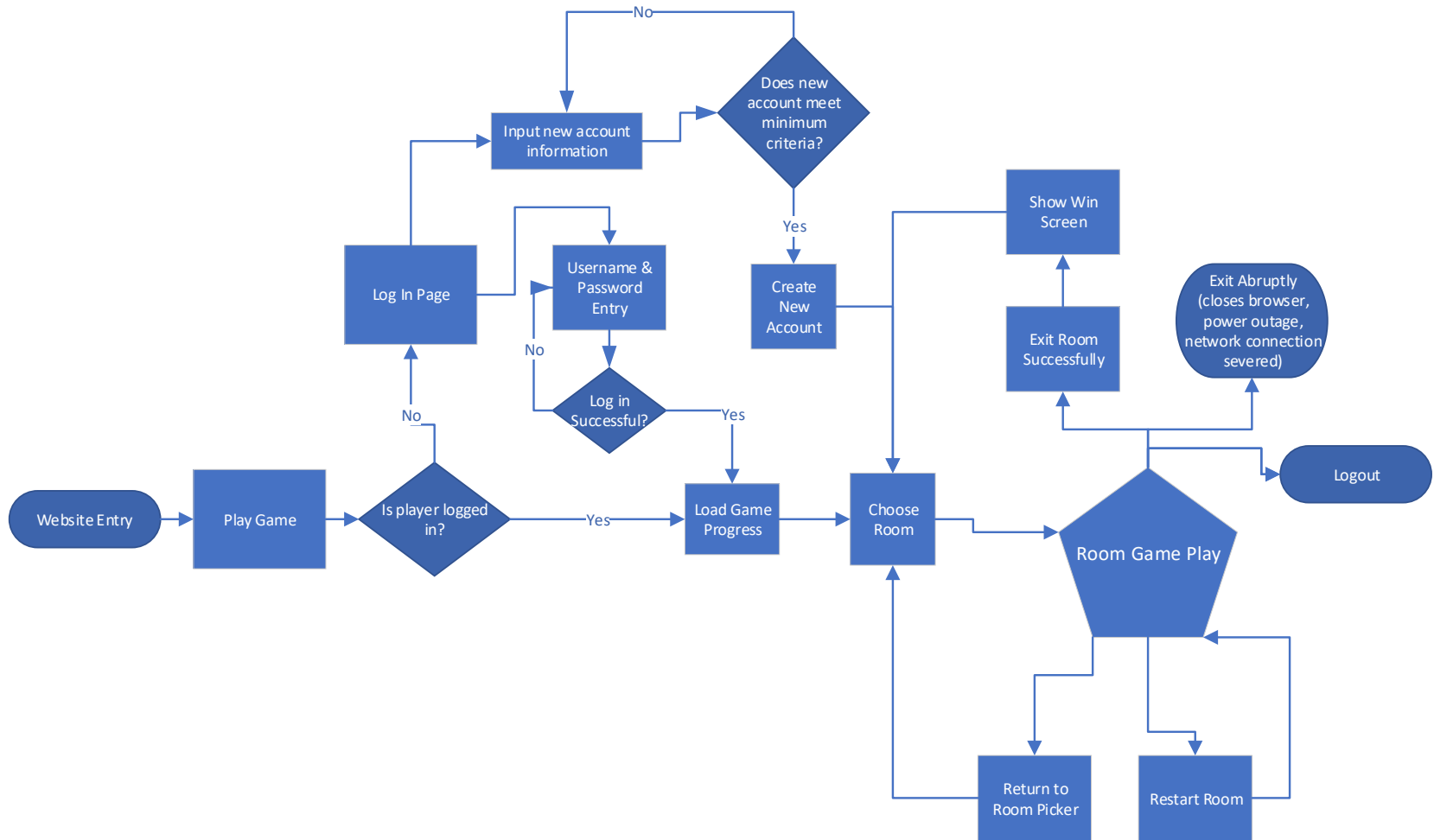


Figure 12. End User Menu Flowchart

Interaction between client and server

Communication is handled in the following steps:

1. **Send command:** The player initiates communication by pressing a key or clicking on a button. The client intercepts the input and sends the command via an XMLHttpRequest method.
2. **Process command:** The server receives the XMLHttpRequest request on the *Play* controller. It retrieves the current game state and pushes the command and the game state deeper into the core game logic to make the appropriate changes. The core logic then sends the updated game state back to the controller.
3. **Send updated game state:** Once the core game logic is complete, the Play controller subsequently transforms the game state for client consumption and returns the game state back to the client in the XMLHttpRequest return value.
4. **Render updated game state:** The client receives the result for the original XMLHttpRequest method, compares the updated game state with the previous game state, and updates the screen in accordance to the new game state. Outdated objects are destroyed. New objects are created and rendered to the screen. If necessary, the entire scene will be switched.

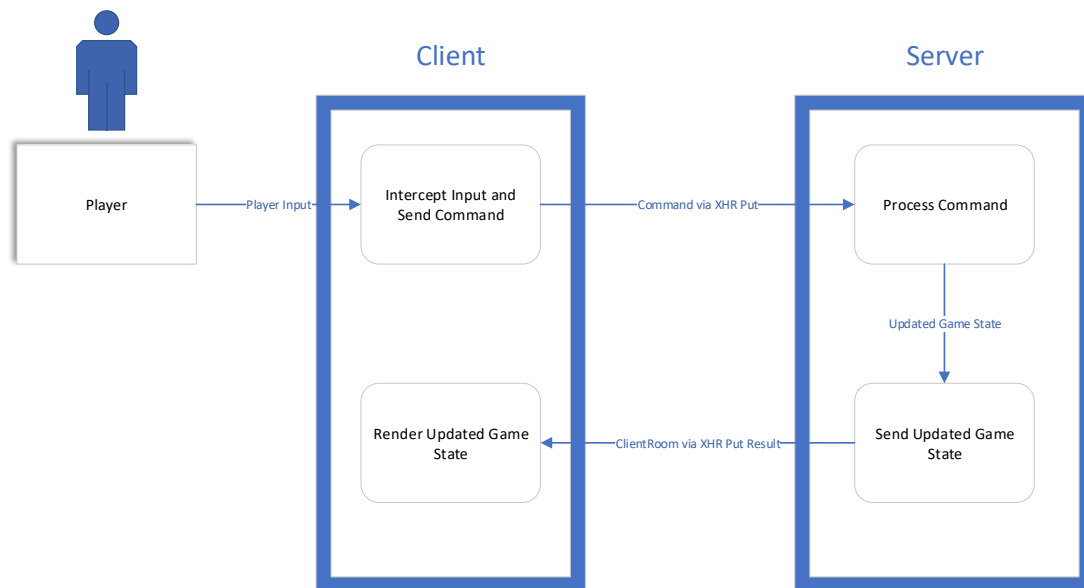


Figure 13. Data Flow Diagram - Client and Server Communication

CHAPTER 4

PROJECT MANAGEMENT

Project Phases

Overall, this project required two phases: a planning phase and an implementation phase.

The planning phase included creating all of the creative content. There already existed some basic requirements prior to the planning phase. The end-result needed to be fun but have some varying level of difficulty. It needed to exercise one's mind. It needed to include the potential for telemetry for future research and enhancements. Beyond the basic idea of the end-result, most of the creative elements of the game were chosen during the planning phase. The basic concept of the game was conjured up during the planning phase. Player motivations and the scoring formula were also devised. Concepts for various levels were created. Graphics and audio were purchased from outside vendors that specialize in game assets. However, no single vendor covered all of the creative requirements so there was a need to carefully select assets that would weave together in a coherent fashion. Other aspects of the game design, such as items and obstacles, were also imagined.

The planning phase also included the creation of all of the user stories to act as the foundation of the requirements. The user stories were based on the overall vision of the game design. They helped to create a clear understanding of the mechanical element of the game.

Once user stories were completed, the next portion of the planning phase included drafting more technical deliverables. These include entity-relationship diagrams, UML class diagrams, sequence diagrams, and wireframes.

One final task during the planning phase was conducting research to select various technologies. It was decided that Microsoft Azure would be the hosting provider, Visual Studio would be the chosen IDE, C# would be the programming language and ASP.NET Core MVC would be the server platform. After carefully researching various game engines, Unity was selected for the game engine and the target platform for the client-side would initially be HTML5 with WebGL.

The implementation phase included putting all of the design into action. This was the phase where all of the software development occurred. Most of the written code is in the core project class library. Entity classes were written incrementally. Interactor classes were also created to interact with the entities. Furthermore, manager classes were also written to assist with class interaction. Finally, data access classes were written.

The implementation phase also included creating the MVC project and integrating the core game logic into the MVC web application. Another project that was written was the unit testing project. As each slice of code was initially prepared, corresponding unit tests were also coded to verify the integrity of the code.

The implementation phase also included debugging, DNS assignment, deployment, and end-user feedback.

Project Management Process

Due to the unpredictability of software development and unknown variables associated with this exploratory and artistic project, a hybrid approach was determined to be implemented in order to see the project to its completion. While the project mostly adhered to a waterfall SDLC approach, there was a need for a more nimble and flexible approach for the implementation phase. Instead of using traditional project management tools (such as a work breakdown structure, PERT charts, and Gantt charts), project management was treated more like an agile project.

During the planning phase, all requirements were gathered and converted into user stories. Users stories used the Connextra template (User Story Template, n.d.) and focused on the end-user experience. Each user story was subsequently given a relative chronological priority and approximate amount of time to develop. All user stories were stored in the product backlog.

The implementation phase was divided into weekly sprints. Each sprint focused on a task-completion workbook similar to a Kanban board. The workbook sheets included: to-do, work-in-progress, and completed. In the beginning of each sprint, user stories were taken from the product backlog and placed on the to-do section of the Kanban workbook. As development proceeded, the user stories were broken down into functional requirements and

tasks to code. By the end of the sprint, all of the user stories would be in the completed section of the Kanban workbook. This cycle continued for the duration of the semester.

User Story List

<i>As A...</i>	<i>I Can...</i>	<i>So That...</i>	<i>Story Points</i>
admin	kick out all players from the system	perform updates and system changes without worrying about players performing any real-time tasks	1
admin	log in to my account from the admin login page	I can securely perform administrative duties	1
admin	reset a player's password if the player's credentials are stored internally	a player has a mechanism of updating their account information just in case they can't log in themselves	1
admin	see a list of established users	I can provide basic administrative duties	1
admin	export all reports to pdf	I can record formatted data or send data to others in pre-formatted layout	1
admin	export all reports to xlsx	I can record data for later, send formatted data to others, or export for further data analytics	1
admin	run a "advancement rate" report that shows how fast each player is advancing through the rooms	I can determine if there's enough difficulty in each room and enough rooms for people to play	1
admin	run a "load" report of how many players are playing by date/time in total	I can assess overall volume of player usage	1
admin	run a "room popularity" report of a list of rooms along with the count of how often they're played	to see which rooms are played the most often	1
admin	run a "user play frequency" report and chart that shows how many days each player has played and how long per day each player has played over a period of time	I can understand game play habits and patterns, such as how long it takes before individual players start losing interest in the game	1
admin	run an "g factor" report that compares per-user difficulty of rooms focusing on memorization and rooms that focus on pattern-matching	to see if there's a correlation in per-user difficulty between rooms that require memory and rooms that require pattern recognition	1

admin	run an "room difficulty" report of a list of rooms along with the count of how much time they take to complete and how often they're reset / given up on	to see which rooms take the longest to complete and require the most resets based on aggregate data	1
admin	see a list of support tickets	I can hope to resolve software bugs	1
admin	broadcast a message to all players currently in the game	they are warned ahead of time if something needs to happen, such as a system update/reset	2
player	click on the "restart room" button while playing in a room	I can try the room again with a fresh new start	1
player	obtain the contents of an unlocked treasure chest by walking over the chest	I can get the contents inside, such as a key or a gem	1
player	add a key into my inventory when I walk over it within the room	I can use that inventory item later in the room	1
player	see a key on the floor unless it is off the screen or hidden by an obstacle	I know where to navigate to get the key in my inventory	1
player	see in the game if I have already picked up a key	I can keep track of my inventory	1
player	browse to the game using a friendly DNS name	I can connect to the site in an easy-to-remember manner	1
player	unlock a locked door with a key	I can enter the previously locked door and win the room	1
player	close my browser and leave the game	I can go for a walk or something	1
player	depress certain floor buttons by walking off of them and undoing whatever effect they had	I face a higher level of difficulty and frustration	1
player	push a floor button by walking over it	the gameplay progresses and I unlock whatever secret it had hidden	1
player	push a movable block over depressable floor buttons to keep the floor button pressed	the effect that the button has is maintained and not undone	1
player	see an uncovered floor button on the floor	I know where to navigate to push the button	1
player	unlock some locked treasure chests by finding a key for them	I can subsequently navigate to the chest and get the treasure inside	1
player	unlock some locked treasure chests by pressing a floor button	I can subsequently navigate to the chest and get the treasure inside	1
player	unlock some locked treasure chests by solving a pattern-matching puzzle within the room	I can subsequently navigate to the chest and get the treasure inside	1
player	click on the "Return to Room Picker" option while playing in a room	I can exit the room and pick a different room to play	1
player	play a simple room that introduces me to the concept and mechanics of moving single-move blocks in isolation to other obstacles (easy room 4)	enjoy learning about the gameplay through playing	1

player	play a simple room that introduces me to the concept and mechanics of moving blocks indefinitely in isolation to other obstacles (easy room 3)	enjoy learning about the gameplay through playing	1
player	see an uncovered treasure chest on the floor	I know where to navigate to get to the chest	1
player	see if a treasure chest is locked or unlocked	I know if getting the treasure will require finding a way to unlock it first	1
player	see if a door is locked or unlocked	I know that winning the room requires finding a key first	1
player	choose to re-enter a room that I have already played	I can try to improve on my previous score	1
player	connect to a webserver and play the game on a modern browser	enjoy playing the game regardless of my underlying platform	1
player	stop moving any direction when I walk up to an obstacle such as a wall or brazier	I obey the laws of physics	1
player	walk through an unlocked door at the end of a room	I beat the room	1
player	type in my established username and password when logging in	restore my progress from the last time I played and continue where I left off	1
player	click on the 'log off' option while playing in a room	I can exit the game	1
player	make sure that my email address isn't already taken	my account is unique	1
player	make sure that my new account's user name isn't already taken	my account is unique	1
player	try to log in again in case my log in wasn't successful	I get another chance just in case I accidentally typed in my credentials incorrectly	1
player	update my account password if my account credentials are stored internally	my account is stays secure	1
player	update my account email address	my account is kept up-to-date per my personal preferences	1
player	see an uncovered gem on the floor	I know where to navigate to get the gem	1
player	stop walking when I run into a brazier	I observe the laws of physics	1
player	add a lantern into my inventory when I walk over it within the room	I can use that inventory item later in the room	1
player	see a lantern on the floor	I know where to navigate to get the lantern in my inventory	1
player	see in the game if I have a lantern in my inventory	I can keep track of my inventory	1
player	not see after the lantern goes out	I face a higher level of difficulty and frustration	1

player	see nothing while in complete darkness	I face difficulty when playing the game	1
player	still technically move while in darkness, I just won't know if I'm successfully making progress towards exiting the room	I face difficulty when playing the game	1
player	temporarily alleviate the darkness completely with a lantern	I can see for a few moments	1
player	click the background music button	optional audio is enabled or disabled to my liking	1
player	choose to load my saved progress when the game first loads	continue where I left off and restore all of my previous progress and accomplishments	1
player	get a recap of how my score in a level after successfully exiting it	know my progress	1
player	see the title screen when I first connect to the game	I am properly introduced to the game	1
player	see how many gems can possibly be obtained in a room	I know if I can still find more and improve my score for the room	1
player	see how many gems I already picked up	I can keep track of my inventory	1
player	see how many times I needed to relight my lantern	my score is adjusted appropriately for the level	1
player	play a simple room that introduces me to the concept and mechanics of dealing with darkness in isolation to other obstacles (easy room 1)	enjoy learning about the gameplay through playing	1
player	play a simple room that introduces me to the concept and mechanics of dealing with illusion walls in isolation (easy room 2)	enjoy learning about the gameplay through playing	1
player	see an illusion wall just like it was a real wall	I face difficulty when playing the game	1
player	walk through an illusion wall	I can progress through the room	1
player	click sound settings for sound options	to see a list of sound options available to player	1
player	click the sound button	optional audio is enabled or disabled to my liking	1
player	push heavy blocks forward to an adjacent empty space once	hidden items, such as gems or keys, are revealed	2
player	push light blocks forward to an adjacent empty space once	hidden items, such as gems or keys, are revealed	2
player	progress forward more slowly as I push blocks forward	I can carefully place the blocks exactly where I need	2
player	connect to the website via https and a valid TLS certificate	I know that my communication is secure	2
player	push any block forward over a floor button	the consequence of the button remains in effect	2

player	choose to play any room from a room picker menu once I successfully log in or create an account	I can start playing	2
player	move up, down, left, and right so long as there isn't anything blocking me	I can navigate around the room to reach the end	2
player	see the view of the room shift alongside my movement as I move around a large room	I can see my general vicinity instead of moving off of the screen	2
player	create a new account when I put in account information that meets the unique criteria	save my progress for later	2
player	link my established account to a Facebook or another established account	I don't need to have a separate account for this game	2
player	start a new account from the main menu	I can begin the account creation process	2
player	request a full GDPR-compliant erase from the "change user account settings" menu option	I am sure that the system does not keep any information on me	2
player	see admin broadcasted messages	I can prepare for any upcoming issues	2
player	pick up hidden gems within a room	I can get extra points for the room after I successfully exit it	2
player	uncover hidden gems by pushing blocks that hide them or unlock chests that hold them	I can see where to navigate to in order to obtain the gems and increase my score	2
player	see a little bit within the proximity of a brazier	I see basic milestones throughout the room	2
player	see braziers and the light that illuminates from them in dark rooms	I can see a little easier within the proximity of the brazier	2
player	(re)light a lantern	I can see where I'm going in a dark room for a short amount of time	2
player	see my surroundings better when I light my lantern	I can navigate more effectively	2
player	see my best score next to my most recent score in a level after successfully exiting it	know my progress	2
player	see my previous best score for each room that I have successfully exited	I can see how well I did and aim to improve my score next time	2
player	play a moderately complicated room that incorporates several obstacles (medium room 1)	Enjoy playing the game with medium difficulty	2
player	play a moderately complicated room that incorporates several obstacles (medium room 2)	Enjoy playing the game with medium difficulty	2
player	play a moderately complicated room that incorporates several obstacles (medium room 3)	Enjoy playing the game with medium difficulty	2
player	play a moderately complicated room that incorporates several obstacles (medium room 4)	Enjoy playing the game with medium difficulty	2
player	play a moderately complicated room that incorporates several obstacles (medium room 5)	Enjoy playing the game with medium difficulty	2

player	push heavy blocks forward to an adjacent empty space once	I can clear a path towards the end of the room	3
player	push heavy blocks forward to an adjacent empty space once	the blocks align in a special pattern to unlock a mystery	3
player	push light blocks forward to an adjacent empty space indefinitely	I can clear a path towards the end of the room	3
player	push light blocks forward to an adjacent empty space indefinitely	the blocks align in a special pattern to unlock a mystery	3
player	play a more complicated room that spans beyond my immediate sight on the screen and incorporates many obstacles (hard room 1)	Enjoy playing with high difficulty	3
player	play a more complicated room that spans beyond my immediate sight on the screen and incorporates many obstacles (hard room 4)	Enjoy playing with high difficulty	3
player	play a more complicated room that spans beyond my immediate sight on the screen and incorporates many obstacles (hard room 5)	Enjoy playing with high difficulty	3
player	choose a room from the room picker that is sorted by difficulty based on previous player experiences	I can intuitively choose a difficulty level that's appropriate for me	3
player	submit a trouble ticket in case the game has some sort of glitch or crashes	I can help alleviate the issue in the future	3
player	play a more complicated room that spans beyond my immediate sight on the screen and incorporates many obstacles (hard room 2)	Enjoy playing with high difficulty	3
player	play a more complicated room that spans beyond my immediate sight on the screen and incorporates many obstacles (hard room 3)	Enjoy playing with high difficulty	3
scheduled task	update the RoomDifficulty table on a regular basis based on data from the Events table	rooms are sorted by difficulty based on actual player experiences	2
scheduled task	periodically delete old and irrelevant database records	the system is less expensive and clogged	2
scheduled task	periodically purge logs	the system is less expensive and clogged	2
scheduled task	connect to a TLS certificate authority and update a soon-to-expire TLS certificate	the site certificate stays current and all connectivity is secure	5
telemetry	record every time a player attempts to move the player's character into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player decides to exit a room and return to the room picker menu option into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player decides to restart a room into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player receives a key into the Events table	appropriate data points are logged to feed reporting needs	1

telemetry	record every time a player starts a room into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player successfully completes a room into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player decides to log off while in a room into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player receives a lantern into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player uncovers and receives a gem into the Events table	appropriate data points are logged to feed reporting needs	1
telemetry	record every time a player uses a lantern into the Events table	appropriate data points are logged to feed reporting needs	1

Table 3. User Story List

CHAPTER 5

CONCLUSIONS

Ultimate outcome

Ultimately, the project was a success. All of the deliverables were created. The design was implemented and it adequately matched the initial vision. The end-result is a product that will entertain and gauge and help develop memory and exercise cognitive skills. It is a fully-functional video game that relies on client/server architecture using industry standards and popular programming paradigms. It also has built-in methods to gather data and accumulate end-user activity. This will be used for further analytics to refine and improve future product enhancements. It could also serve as a foundation for additional research.

What was learned

We learned a substantial number of topics through this project. This entire project was practically an exploratory project with new technologies. Generally speaking, we learned how to manage a creative technical project in an iterative fashion. While much of the development was agile in nature, the artistic side of the project was fleshed out completely ahead of time. The tasks required to contribute an artistic component were fundamentally different than the tasks required to create the technical system design.

This project introduced us to Microsoft Azure as a hosting provider. Microsoft Azure provides a plethora of services, but the entire process was a bit overwhelming at first. We ended up choosing an app service to host the MVC and class library projects. We also created a separate SQL server resource. Both the app service and server resource exist within a single created resource group.

Prior to this project, we had no first-hand experience with the Unity game engine. This project inspired us to research several game engines. Ultimately, Unity was selected for this project. It provided a rich toolset for game development. Unity provided a point-and-click integrated development environment for easily creating a game layout. It also allowed

for powerful scripting using C#. That being said, the IDE was not completely intuitive and the C# target platform was Mono, distinct from ASP.Net Core. The API was also unique to Unity. This gave Unity a learning curve that required studying. A few months to learning Unity was necessary in order to be productive.

Obstacles

Creative aspects of game

One obstacle faced was simply how to apply creativity to project management. Creating a time estimate for tasks that required subjectivity seemed more difficult. For example, each room created was represented by one user story. Each user story had an estimate time cost. However, we had no experience with accurately estimating the time required to make something difficult and fun.

Additionally, we struggled to find a method to objectively gauge the entertainment dimension associated with Escape Puzzler. How does one create a technical specification and measurement of “fun”?

We also faced difficulty mixing and matching all of the game assets to make a united coherent finished product. If this project had to start over again, recruiting an art major to assist with custom graphics instead of stock graphics would be advisable.

Mixing Unity with ASP.Net Core MVC

Mixing Unity and ASP.NET Core MVC posed a few extra challenges. For example, one preferred method of networking within the Unity does not work if HTML/WebGL is the target platform. For now, the application resorts to implementing XHR requests. In the future, WebSockets might be considered. Another issue faced was the MIME types of the Unity compiled output. In order to generate output to the HTML/WebGL platform, Unity first transpiles the C# code into C++ using a program called, “IL2CPP”. From there, it turns the C++ code into WebAssembly. For some unknown reason, the output extension that it uses is .UNITYWEB instead of something more standard (such as .WASM). In order for the web server to properly serve the outputted files, the following was added to the MVC Startup Configure method:

```
// Unity-specific web assemblies.
var provider = new FileExtensionContentTypeProvider();
provider.Mappings[".unityweb"] = "application/octet-stream";
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "Build")),
    RequestPath = "/Build",
    ContentTypeProvider = provider
});
```

Azure SQL not having SSRS

During the design phase, Azure SQL was selected as a technology for the RDBMS. We were under the false assumption that it functioned similar to its on-premise counterpart, Microsoft SQL Server. It was expected to have a component called, “SQL Server Reporting Services”. Several user stories were written taking that into consideration. During the implementation phase we discovered that Azure SQL did not have SQL Server Reporting Services, nor did it offer any equivalent service. Due to limited time constraints and the fact that the reports were fairly inconsequential to the core goals of the project, we decided to abandon the user stories related to reports for the moment.

Different Plane Geometry

Through trial and error, we discovered that Unity uses the standard two-dimensional Cartesian coordinate system. However, we expected the system to align with a two-dimensional array instead. We expected the origin (0, 0) to be in the top-right of the screen and the bottom-right of the screen to be the highest possible number. All of the multi-dimensional arrays on the server relied on the element index numbers to dictate their geographic location. To accommodate this difference, a simple conversion on the client converts the element index number into something visually similar to quadrant IV. All that was required was to multiple the Y-coordinate array index by -1 on the client.

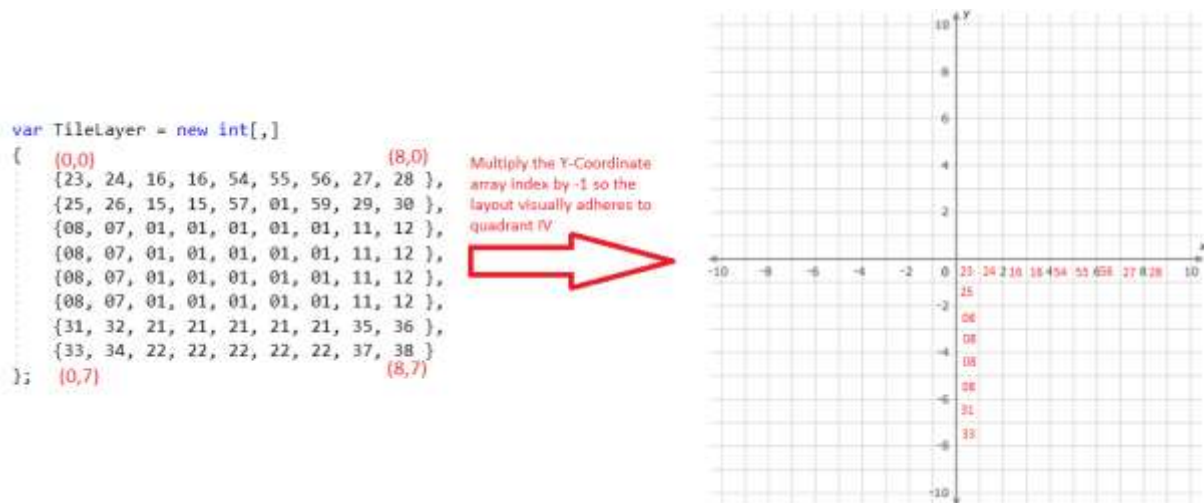


Figure 14. Negate y-coord to align multidimensional array to quadrant IV

AJAX latency

The XHR requests between the client and the server are carefully serialized. When the client sends a request and the server returns the updated game state, further communication is disabled until the updated game state has been completely rendered. For example, the player character might walk to the new destination. Once complete, the communication can resume. Once the app deployed on to Azure, a small (but noticeable) amount of latency surfaces on occasion between the client and the server. For example, when the player presses a directional key, a few milliseconds pass prior to any movement. Furthermore, sometimes the player character might stop walking between movements from tile to tile. This issue might be trivial, but it might also become annoying to a player. To resolve the issue, we are considering promoting the AJAX controller methods to WebSocket connections. This might help with a tighter synchronization.

Next Steps

This project may just be the beginning. With this project's completion, many doors are now open for new opportunities. First, this project is essentially a game made for enjoyment. As such, end-user testing and feedback is essential for improvement. The first steps for an additional phase should take end-user feedback into consideration.

Secondly, this project may act as a stepping stone for further end-user analysis. Now that the foundation is laid, there exists a possibility to adapt this solution into a mechanism for acquiring data for additional research. Such research might include psychometrics, such as correlating an individual's intelligence quotient with their ability to solve puzzles or providing quantitative data for testing the multiple intelligence theory versus the general intelligence theory.

Plan for ADA compliance

This web app has not been thoroughly tested for extended use for individuals with disabilities. Like any web-based technology, Escape Puzzler should strive for a higher level of ADA compliance. Multiple forms of player input can be accepted. Alternative colors can be used for the tile set. The rendered output should be resizable. These things provide several advantages. They potentially increase the target audience. They also improve SEO efforts. Additionally, they may help with overall website usability and make the game more adaptable for a general audience.

Add more rooms

Escape Puzzler is currently in a completed state, but many of the levels merely showcase the mechanical elements of the game. More rooms can be added to for replay value and allow the player to further immerse in the game.

REFERENCES

- (n.d.). Retrieved April 2019, from Unity Learn Premium: <https://learn.unity.com>
- Armstrong, T., & . (2018). *Multiple Intelligences in the Classroom*. Alexandria: ASCD.
- Avgustova, A. (n.d.). *Loot icons*. Retrieved April 2019, from GameDev Marketplace:
<https://www.gamedevmarket.net/asset/loot-icons-8786/>
- Cohen, D. (2016, August 19). *Facebook Developing New PC Gaming Platform; Teams Up With Unity Technologies*. Retrieved from Adweek:
<https://www.adweek.com/digital/facebook-developing-pc-gaming-platform-unity-technologies/>
- Edwardsen, F., & Kulle, H. (2010). *Educational Games: Design, Learning and Applications*. New York: Nova Science Publishers, Inc.
- Elliott, J. L. (2013). *HTML5 Game Development with GameMaker*. Birmingham: Packt Publishing.
- Fisher, C. (2015). *Designing Games for Children: Developmental, Usability, and Design Considerations for Making Games for Kids*. New York: Focal Press.
- Freeman, A. (2017). *Pro ASP.Net Core MVC 2*. New York: Apress.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.
- Hodges, J. (2012, November). *HTTP Strict Transport Security (HSTS)*. Retrieved from IETF RFC: <https://tools.ietf.org/html/rfc6797>
- Jeffrey, C. (2017, May 18). *Microsoft and Unity team up to offer \$100,000 to the best AR design idea*. Retrieved from Tech Spot: <https://www.techspot.com/news/69380-microsoft-unity-team-up-offer-100000-best-ar.html>
- Lander, R. (2019, May 6). *Introducing .NET 5*. Retrieved from Microsoft DevBlogs:
<https://devblogs.microsoft.com/dotnet/introducing-net-5/>
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River: Prentice Hall.
- National Research Council. (2011). *Learning Science Through Computer Games and Simulations*. (M. A. Honey, & M. L. Hilton, Eds.) Washington: National Academies Press.

Rogers, S. (2014). *Level Up! The Guide to Great Video Game Design*. West Sussex: Wiley.

Scolastici, C., & Nolte, D. (2013). *Mobile Game Design Essentials*. Birmingham: Packt Publishing.

Somasegar, S. (2015, April 17). *Visual Studio for Game Development: New Partnerships with Unity, Unreal Engine and Cocos2d*. Retrieved from Microsoft DevBlogs:
<https://devblogs.microsoft.com/somasegar/visual-studio-for-game-development-new-partnerships-with-unity-unreal-engine-and-cocos2d/>

Thomas, D. (n.d.). *Daniel Thomas Asset Store*. Retrieved May 2019, from Unity Asset Store:
<https://assetstore.unity.com/publishers/15413>

User Story Template. (n.d.). Retrieved from Agile Alliance:
<https://www.agilealliance.org/glossary/user-story-template/>

WebGL Browser Compatibility. (2018, September 14). Retrieved from Unity Documentation:
<https://docs.unity3d.com/Manual/webgl-browsercompatibility.html>

APPENDICES

APPENDIX A: BROWSER COMPATIBILITY

The following is a list of compatible browsers with the Escape Puzzler. Other browsers might also work, but these are the one's that have been confirmed to work with Unity's WebGL output. (WebGL Browser Compatibility, 2018):

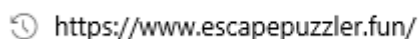
<i>Browser</i>	<i>Version</i>
Mozilla Firefox	67
Google Chrome	75
Apple Safari	12.1
MS Edge	18

Table 4. Browser Compatibility

APPENDIX B: USER MANUAL


Many of the user-specific components that would be inserted in a user manual are already covered in the game design section of the system design chapter. For brevity, that information is not duplicated here. The content includes playable objects, tiles, the scoring system, and obstacles. However, there were a few components applicable to a user manual not previously covered. They are covered in this appendix.

In a compatible browser, navigate to the Escape Puzzler website:



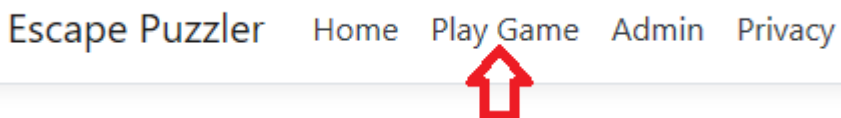
https://www.escapepuzzler.fun/

The main menu exists in the top-left of the web site:



Escape Puzzler Home Play Game Admin Privacy

Click on “Play Game” to begin play:



Escape Puzzler Home Play Game Admin Privacy

If you’re not already logged in, then the website will prompt you to log in. Use your email address and password. If you would prefer to use your Facebook credentials to log in, you can click on the “Facebook” button on the right side.

Use another service to log in.



To create a new account, simply select the “Register as a new user” option.

Log in

Use a local account to log in.

Email

Password

Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)



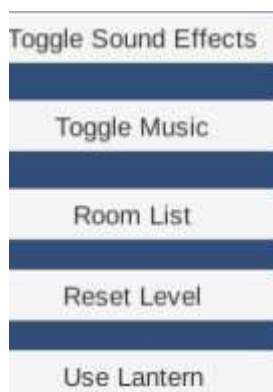
Once logged in, gameplay will begin. The game starts with a start screen. Press the “Start” button to get to the room list.



This brings up the room list. You can now choose a room to play.



During gameplay, the user menu is located in the bottom-right of the game screen:



<i>Menu Option</i>	<i>Usage</i>
Toggle Sound Effects	Turns sound effects on or off
Toggle Music	Turns the background music on or off
Room List	Abandons the current room and goes back to the room list
Reset Level	Starts the room over back in its initial state
Use Lantern	If the player has acquired a lantern in a dark room, this option allows the player to use the lantern and lighten up the room

Table 5. Menu Options

Beyond these instructions, all of the end-user information should be contained in the game design section.

APPENDIX C: LICENSE AGREEMENTS

All of the purchased game assets for the Escape Puzzler project use the standard GameDev market license, with one exception. The exception is the sprite sheet for the main player character. Those graphics fall under the license from Toke Game Art.

Toke Game Art

[\(http://tokegameart.net/licenses/\)](http://tokegameart.net/licenses/)

License for Paid Art

Purchasing art from this site grants you a non-exclusive, non-transferrable license to use the art.

You can NOT resell the art source files (PNG, JPG, EPS, Adobe Illustrator, etc) or slightly modified version of the art. You can not redistribute the art or modified version of the art in a manner that would make some or all of the art files useable to another end user via the app. For example, an app that uses the art as part of the play of the game is fine. An app that allows the user to save or export a modified version of the artwork itself is not fine.

You may use the art in a template of any nature for distribution or sale to third parties.

You may use the art for single game/app title for each platform.

You may not upload the original art files on a website in a complete or archived downloadable format that would make them accessible to others. Backing up to your personal Dropbox, Google Drive, etc is fine.

The license is valid for entities with annual revenue of less than \$500,000 USD or equivalent foreign currency. If annual revenues exceed this amount at any time, the license is no longer valid and you will have to contact us for an Extended License or all use of the art and distribution must cease.

There are no refunds on digital products once you have downloaded them on your system.

Our paid licenses are specifically intended for use in digital media, such as an online game, mobile app, PC game, etc. Using the art in promotional materials for the app or game is allowed, for example, as part of the App Icon or Featured Images of the app. If you intend to use the work in vastly different way, please get in touch with tokegameart. For example, if you are writing a programming book and need graphics for print, that would require a different licensing agreement. If you aren't sure of something that is not addressed explicitly in this license, assume it is not allowed and please contact us via support for clarification and permission.

Regarding ownership of the art: the artist (licensor) owns all proprietary rights in and to all copyrightable works and has the exclusive right to license others to produce, copy, make, or sell their art. Plainly speaking, the artist is the copyright holder, but you are granted a license to use the art in your app or game based on the terms above.

Your app does not require attribution or credit to the Tokegameart. But feel free if you'd like to.

Standard GameDev Market License

<https://static.gamedevmarket.net/terms-conditions/#pro-licence>

LICENCE (A) – For purchases made after 00:00 (GMT) on 15th January 2019

4.1. A “Licence” means that the Seller grants to GDN (purely for the purpose of sub-licensing to the Purchaser) and GDN grants (by way of sub-licence thereof) to the Purchaser a non-exclusive perpetual licence to;

(a) use the Licensed Asset to create Derivative Works; and

(b) use the Licensed Asset and any Derivative Works as part of both Non-Monetized Media Products and Monetized Media Products, with no restriction on the number of projects the Licensed Asset may be used in. In either case, the Licensed Assets can be used in Media Products that are either:

i) used for the Purchaser's own personal use; and/or

ii) used for the Purchaser's commercial use in which case it may be distributed, sold and supplied by the Purchaser for any fee that the Purchaser may determine.

4.2. A Licence does not allow the Purchaser to:

(a) Use the Licensed Asset or Derivative Works in a logo, trademark or service mark;

(b) Use, sell, share, transfer, give away, sublicense or redistribute the Licensed Asset or Derivate Works other than as part of the relevant Media Product; or

(c) Allow the user of the Media Product to extract the Licensed Asset or Derivative Works and use them outside of the relevant Media Product.