Prison Break Final Report

Fall 2020

Group 12: Jacob DiStefano, Dan Hrubec, Julian Gonzales, John Mistica

Table of Contents

List of Figures

I Project Description

1 Project Overview

Prison break is a simple game where you try to escape a prison. You must traverse through multiple different rooms while solving puzzles, and escaping guards. You can collect power ups as well as other items that will aid in your escape. The guards will be exploring the rooms to try and find you, where you lose if they manage to catch you.

2 Project Domain

The project was created using the Unity engine. Most of the development of the project was coded using C# script files to manage the flow of the game. Testing of the game will mostly be done by physical testing of the games elements. So we will run the game and test all the scenarios of that aspect of the game to ensure that our script files handle all cases of interaction.

3 Relationship to Other Documents

The project idea and concept is detailed in the final report from Group 9 in Spring 2019 [1]. The development progression of the project is documented via the Scenario I and Scenario II documents. In those documents we outline the goals we needed to reach during each of their respective sprints. It also describes some of the deliverables of the game and what to expect. Other additions to the game that were not listed in the Scenario documents are recorded and logged via Jira.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Prisoner: The character that you are controlling. The character wears an orange jumpsuit and is attempting to escape the prison.

Guard: The prison guard wearing the built uniform. Guard NPCs roam around the room attempting to find the prisoner and will chase them if they come close.

Item: Interchangeable with the word powerup. Items are scattered around the room and can be picked up once coming into contact with them. Once picked up they will appear in the inventory.

Inventory: Appears on the bottom left of the game screen during gameplay. It tracks the items that the player picks up. Players can cycle through each inventory slot which is highlighted and can use them at their free will.

Key: Special items that appear in some of the rooms. The door will appear locked if the player has not used the Key. Players cannot enter into the next room unless they use the key.

Door: Doors appear somewhere in each level. Interacting with the door by coming into contact with it will send the player to the next room. Unless the room requires a key which will need to be used before the door can be interacted with.

Room: Describes the current level the player is in. Each room is encased around walls and players need to progress through each room to reach the end and ultimately win the game.

Walls: Walls are borders around a room, which act as boundaries for the player. The player cannot walk past the walls and must stay within the walls for each room.

Controls: Controls refer to the keys used for the game. The WASD keys denote the movement in the game, ESC brings up a pause menu, U cycles through the inventory slots and I uses the item in the highlighted slot.

4b UML and Other Notation Used in This Document

The only notation used is UML. It follows the standard guidelines described in “UML Distilled” by Fowler.

4c Data Dictionary for Any Included Models

Levels are generated via .csv files. It takes in the dimensions of the room, the amount of guards, the locations of the guards, and what type of tile is at each coordinate in the grid. If no values are specified, it loads a default floor tile.

Sprites are loaded in via .png files from Sprite sheets. Sections of the sprite sheets are loaded in game as items, guards, and tiles.

II Project Deliverables

The development of *Prison Break* was split into two releases, each of which consisted of four sprints worth of development. For the purposes of this project, each sprint was one week, so each release was four weeks apart.

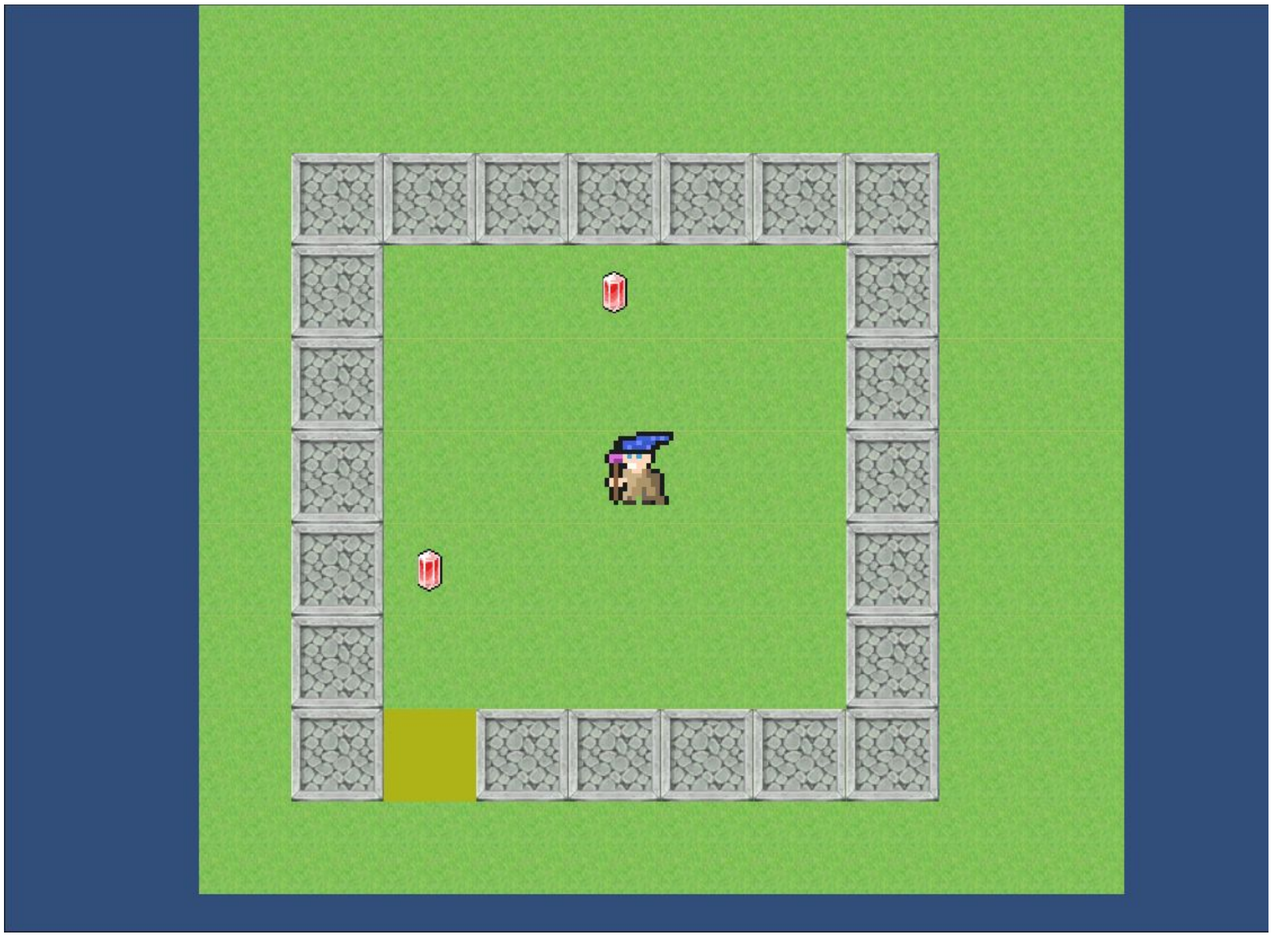
5 First Release

The first sprint was dedicated to getting the infrastructure for the project in place. This included general project initialization, such as getting the .gitignore file in order and the Unity Project set up so all team members could build it.

The next sprint was dedicated to the User Interface and Map Creation functionalities. We placed emphasis on the map creation and storage in order to simplify map creation further down the line, and have infrastructure in place for storing maps when the need arose.

The third and fourth sprints were dedicated to player creation and movement, as well as item creation and interactions. These were both essential to being able to play the game, and we wanted to have a generic interface for spawning players and items. We chose to save custom art for the project until the second release, to ensure that the base functionality of the release was complete.

The combined development from each of these sprints made up the first release, pictured below. All of the promised objectives from the planning stage were included in the release.



6 Second Release

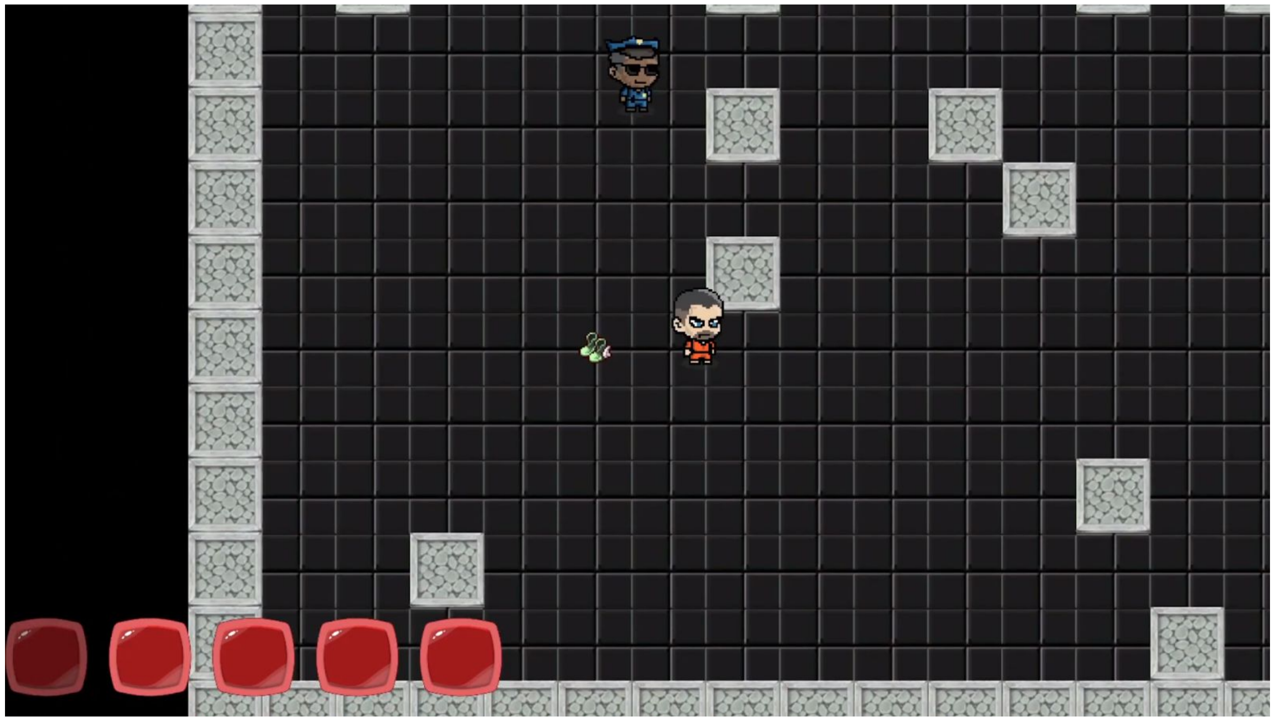
The second release was decided to be much more ambitious than the first. We planned out a multi room demo for the project, which would act as a vertical slice of the potential game, showcasing all possible features.

The first sprint was dedicated to supporting multiroom progression, to allow for the full six room demonstration. This mostly involved creating doors to switch levels, and using the map storage utility to load in a new room. We also used this sprint to begin adding enemies (guards) onto the map as part of the demonstration. This sprint was also used to fix the art style, which was previously using placeholders images.

The second sprint was dedicated to getting guards to move around and track the player, as well as creating the interface for choosing and using items. The guard AI was particularly important to our demo, as the game would be trivial to play if the guards did not track or search for the player.

The third sprint was dedicated to designing the remaining rooms for the demo. This included rooms that used locks and keys to prevent the player from progressing, and implementing a text input interface to support puzzles. This sprint was also used to develop a failure and victory state, for whenever the player successfully navigated the demo.

The final sprint was used for polishing the demo and fixing any remaining bugs, such as guards spawning or moving outside the map or into walls. After the remaining polish was complete, the project reached its second release. A screenshot from this release is pictured below.



7 Comparison with Original Project Design Document

This project was initially proposed by Group 9 from the Spring 2019 semester [1]. Several features from their description were incorporated into our demo implementation, but many were cut as possible future features for a full fledged implementation.

Our implementation of the project did faithfully represent many aspects from the original proposal for the project by Group 9. The original project description dictated that the game would be a map based prison break game where characters, guards and items would have coordinates logged by the system. Our implementation successfully honors this, especially in regards to the map storage and creation interface, which works based off a tiled coordinate system.

However our implementation was not able to capture all elements of the original proposal. Multiplayer was a large element in the original proposal, with players coordinating with one another in order to skillfully evade the guards. Also the proposal stated that players should be able to pick a different character upon each play session, however our implementation only had one character.

III Testing

8 Items to be Tested

The following functionalities will be tested, with them are listed the primary responsible source code files

Inventory and Items: Item.cs, ItemFactory.cs, Inventory.cs

Map Storage/Loading: GridScript.cs

Guard AI: GuardPathing.cs

Player Movement/Collisions: MoveCharacter.cs

9 Test Specifications

ID# - FT1

Description: Testing interactions with an item, it will add it to the inventory.

Items covered by this test: Items, Inventory

Requirements addressed by this test: If a player interacts with an item with a it should add it to the inventory.

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: Loading into the game, we can move our character over and interact with an item.

Input Specification: The input will depend on the players gameplay throughout the game. Specifically, if the player interacts with items scattered throughout the map.

Output Specifications: The item should appear in the inventory in the next available slot.

Pass/Fail Criteria: The test will pass if the player can interact with an item and it is added to the inventory. Otherwise, the test will fail.

ID# - FT2

Description: Testing interactions when the inventory is full.

Items covered by this test: Items, Inventory

Requirements addressed by this test: If a player interacts with an item with a full inventory, the item should not be added to the inventory, and the item will not disappear from the map.

Environmental needs: N/A

Intercase Dependencies: FT1

Test Procedures: In the current demo of the game, there exists more than 5 items in the last room. If we fill up our inventory and interact with another item, the test should be set up.

Input Specification: The input will depend on the players gameplay throughout the game.

Output Specifications: The output should notify the player has a full inventory, and the item should remain on the map.

Pass/Fail Criteria: The test will pass if the player can interact with an item with a full inventory. If they do, the item will not appear in the inventory, and the item will still appear on the map. Otherwise, the test will fail.

ID# - FT3

Description: Testing interactions when using an item.

Items covered by this test: Items, Inventory

Requirements addressed by this test: If the player uses an item, it is removed from the inventory.

Environmental needs: N/A

Intercase Dependencies: FT1

Test Procedures: When we have an item in our inventory, we can use the U key to use the item at the highlighted inventory slot.

Input Specification: The input will depend on the players gameplay throughout the game. Specifically when the player has an item in their inventory.

Output Specifications: The output should be that the item is removed from the inventory UI.

Pass/Fail Criteria: The test will pass if the player can successfully use an item, and it is removed from their inventory. Otherwise, the test will fail.

<u>ID# - FT4</u>

Description: Testing interactions when we use an item.

Items covered by this test: Items, Inventory

Requirements addressed by this test: If the player uses an item, the respective interaction with the item is executed. For example, if the item is a key, it will unlock the door, and if the item is a power up, the respective boost will be given.

Environmental needs: N/A

Intercase Dependencies: FT1, FT3

Test Procedures: In the game, there are interactions with a key item and speed boost items. Interacting with these items should unlock the door, and give the player more movement speed respectively.

Input Specification: The input will depend on the players gameplay throughout the game. Specifically when using a powerup item or a key item.

Output Specifications: The output depends on the item used. If the item is a speed boost powerup, the player should have more movement speed, if the item is a key, the door should be unlocked.

Pass/Fail Criteria: The test will pass if the players interactions with items match with the output. Otherwise, the test will fail.

ID# - FT5

Description: Testing interactions when we use an item, but we do not have anything in the inventory.

Items covered by this test: Items, Inventory

Requirements addressed by this test: If the player attempts to use an item, but the highlighted inventory slot is empty, nothing should happen, the game should continue as usual.

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: When initially loading into the game, the inventory is empty. Attempting to interact with an empty inventory should not crash the game, and the game should continue as normal.

Input Specification: The input will depend on the players gameplay throughout the game, but can be executed when loading into the game.

Output Specifications: The expected output is that no changes to the game will be made.

Pass/Fail Criteria: The test will pass if the player can attempt to use an item on an empty inventory slot, but nothing will happen.

ID# - FT6

Description: Given an invalid map file, the game should exit

Items covered by this test: Map Storage, Map Loading

Requirements addressed by this test: If a developer creates an invalid map file, the game should reject it and not show the player a broken map

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: When loading *Test1Scene.unity* the game should not show anything, indicating that the game has not loaded anything

Input Specification: Input is a map .csv file with a map size smaller than the coordinates of tiles placed on it

Output Specifications: The expected output is no map to be loaded

Pass/Fail Criteria: If the game crashes (gracefully or otherwise) the test passes, if the player is able to enter the map, the test fails.

ID# - FT7

Description: Test normal map loading procedures, confirming that all tiles can successfully load during normal execution

Items covered by this test: Map Loading

Requirements addressed by this test: The game should be able to load in a map from a storage file. The game should be able to load and render any type of predefined tile

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: When loading this test the player should be spawned into a room with walls and every available item, along with a door.

Input Specification: The input is a valid .csv map file of a reasonable size including walls, at least one of every item, and a door

Output Specifications: The expected output is no errors, and a successful load

Pass/Fail Criteria: The test will pass if the tester is able to see every tile and map rendered correctly, otherwise the test fails.

ID# - FT8

Description: Testing the boundary condition of a very small map

Items covered by this test: Map Loading

Requirements addressed by this test: Any valid map file should be successfully loaded

Environmental needs: N/A

Intercase Dependencies: FT7

Test Procedures: When loading a valid .csv map file with a very small size (height and width < 3), the game should successfully load with no errors

Input Specification: The input is a valid .csv file with height and width less than 3

Output Specifications: The expected output is a successful load with no errors

Pass/Fail Criteria: The test will pass if the tester is able to see every tile and map rendered correctly, otherwise the test fails.

ID# - FT9

Description: Test the loading of a very large .csv map file (height and width > 10,000), the game should successfully load with no errors

Items covered by this test: Map Storage

Requirements addressed by this test: Any valid map file should be successfully loaded

Environmental needs: N/A

Intercase Dependencies: FT7

Test Procedures: When loading a valid .csv map file with a very large size (height and width > 10,000), the game should successfully load with no errors

Input Specification: The input is a valid .csv file with height and width greater than 10,000

Output Specifications: The expected output is a successful load with no errors

Pass/Fail Criteria: The test will pass if the tester is able to see every tile and map rendered correctly, otherwise the test fails.

ID# - FT10

Description: Test normal map loading procedures with NPCs, confirming that all tiles can successfully load during normal execution

Items covered by this test: Map Loading

Requirements addressed by this test: The game should be able to load in a map from a storage file. The game should be able to load and render any type of predefined tile and NPC

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: When loading this test the player should be spawned into a room with walls and every available item, along with a door and multiple NPCs

Input Specification: The input is a valid .csv map file of a reasonable size including walls, at least one of every item, and a door and at least two NPCs

Output Specifications: The expected output is no errors, and a successful load

Pass/Fail Criteria: The test will pass if the tester is able to see every tile and NPC with the map rendered correctly, otherwise the test fails.

ID# - FT11

Description: Testing behavior of AI to see if the guard will follow the player once getting within a close proximity of the player.

Items covered by this test: AI

Requirements addressed by this test: The guard should be able to detect and follow the player once the player gets within a certain radius of the guard.

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: The player must walk close enough to the guard for the AI to detect the player and then follow the player.

Input Specification: The input will depend on how the player plays the game and how they navigate through the rooms specifically if they get close enough to a guard.

Output Specifications: The expected output is that guard being able to follow the player

Pass/Fail Criteria: The test will pass if the guard is able to follow an detect the player once the player gets within a certain radius of the guard.

ID# - FT12

Description: Testing AI to see if it will stop following the player once the player leaves the detection proximity of the guard.

Items covered by this test: AI

Requirements addressed by this test: If the player is far enough away from the guard they will stop following the player.

Environmental needs: N/A

Intercase Dependencies: FT11, FT4

Test Procedures: The player must be able to pick up and use a speed boost item and then outrun the guard.

Input Specification: The input is the player picking up and consuming the speed boost and outrunning the guard.

Output Specifications: The expected output is that the guard will stop following the player.

Pass/Fail Criteria: The test will pass if the guard stops following the player once the player has outrun and left the detection radius of the guard.

ID# - FT13

Description: Testing AI to see if it will stop following the player once losing a direct line of sight with the player.

Items covered by this test: AI

Requirements addressed by this test: The AI should be able to stop following the player if they lose direct line of sight with the player for example if the player was hiding behind a wall.

Environmental needs: N/A

Intercase Dependencies: FT11, FT10

Test Procedures: The player must first lure a guard and then proceed to hide behind a wall.

Input Specification: The input is the dependent on the how the player plays the game, but the player must first lure a guard and then find a wall tile to hide behind.

Output Specifications: The expected output is the guard will stop following the player.

Pass/Fail Criteria: The test will pass if the guard stops following the player once the player gets behind a wall tile and loses direct line of sight with the guard.

ID# - FT14

Description: Testing AI to see if the guard will not follow or detect the player if the player stays a large distance away from the guard.

Items covered by this test: AI

Requirements addressed by this test: The AI should not find or detect a player if the player stays a far enough distance away from the guard.

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures: The player should keep a large enough distance between them and the guard so that the guard does not detect the player.

Input Specification: The input is dependent on how the player plays the game, but the player must stay a large enough distance away from the guard.

Output Specifications: The guard will not detect or follow the player.

Pass/Fail Criteria: The test will pass if the guard does no detect or follow the player if the player is far enough away from the guard.

ID# - FT15

Description: Testing AI to see if multiple guards can follow the player at once

Items covered by this test: AI

Requirements addressed by this test: The AI should be able to still detect and follow the player if there are multiple guards pursuing the player.

Environmental needs: N/A

Intercase Dependencies: FT11

Test Procedures: When loading this test the player should get within detection distance of multiple guards and have multiple of them follow the player at once.

Input Specification: The input is dependent on how the player plays the game, but the player should purposely get close enough to multiple guards to have them follow the player.

Output Specifications: The expected output is that multiple guards will be able to follow the player at once.

Pass/Fail Criteria: The test will pass if multiple guards are able to follow the player at once without losing sight of the player.

10 Test Results

ID# - FT1

Date(s) of Execution: 11/23/2020

Staff conducting tests: Dan Hrubec

Expected Results: The item will be added to the inventory.

Actual Results: The item was added to the next available inventory slot.

Test Status: Pass

ID# - FT2

Date(s) of Execution: 11/23/2020

Staff conducting tests: Dan Hrubec

Expected Results: The item will be able to be interacted with, but it will be removed from the map.

Actual Results: The item was able to be interacted with, but it did get removed from the map.

Test Status: Fail

ID# - FT3

Date(s) of Execution: 11/23/2020

Staff conducting tests: Dan Hrubec

Expected Results: Using an item will remove it from the inventory.

Actual Results: The item was successfully removed from the inventory.

Test Status: Pass

ID# - FT4

Date(s) of Execution: 11/23/2020

Staff conducting tests: Dan Hrubec

Expected Results: The item will be added to the inventory.

Actual Results: The item was added to the next available inventory slot.

Test Status: Pass

ID# - FT5

Date(s) of Execution: 11/23/2020

Staff conducting tests: Dan Hrubec

Expected Results: The respective interactions with the items will be successful.

Actual Results: Speed boosts properly applied, keys unlock the doors.

Test Status: Pass

ID# - FT6

Date(s) of Execution: 11/24/2020

Staff conducting tests: Jacob DiStefano

Expected Results: The map would fail to load, either crashing the game or outputting an error

Actual Results: The map loaded successfully, with no errors

Test Status: Fail

ID# - FT7

Date(s) of Execution: 11/24/2020

Staff conducting tests: Jacob DiStefano

Expected Results: The map would load successfully with all items and tiles rendered

Actual Results: The map was loaded successfully with all items and tiles rendered

Test Status: Pass

ID# - FT8

Date(s) of Execution: 11/24/2020

Staff conducting tests: Jacob DiStefano

Expected Results: The map would load successfully with all items and tiles rendered

Actual Results: The map was loaded successfully with all items and tiles rendered

Test Status: Pass

ID# - FT9

Staff conducting tests: Jacob DiStefano

Expected Results: The map would load successfully with all items and tiles rendered

Actual Results: The map did not load successfully, and actually crashed the tester's computer

Test Status: Fail

<u>ID# - FT10</u>

Date(s) of Execution: 11/24/2020

Staff conducting tests: Jacob DiStefano

Expected Results: The map would load successfully with all items, tiles and NPCs rendered

Actual Results: The map was loaded successfully with all items, tiles and NPCs rendered

Test Status: Pass

<u>ID# - FT11</u>

Date(s) of Execution: 11/25/2020

Staff conducting tests: John Mistica

Expected Results: Guard will follow the player once the player gets within close proximity of guard.

Actual Results: Guard successfully followed the player once they got within a close proximity of the guard.

Test Status: Pass

<u>ID# - FT12</u>

Date(s) of Execution: 11/25/2020

Staff conducting tests: John Mistica

Expected Results: While already following the player, the guard will stop following once the player distances themselves from the guard.

Actual Results: Guard successfully stopped following the player once they distanced themselves from the guard.

Test Status: Pass

<u>ID# - FT13</u>

Date(s) of Execution: 11/25/2020

Staff conducting tests: John Mistica

Expected Results: Guard will stop following the player once they have lost a direct line of sight of with the player.

Actual Results: Guard failed to stop following the player once the player got behind a wall.

Test Status: Fail

<u>ID# - FT14</u>

Date(s) of Execution: 11/25/2020

Staff conducting tests: John Mistica

Expected Results: Guard will not follow the player if the player keeps a substantial distance between them and the guard.

Actual Results: Guard successfully failed to follow the player while the player kept a substantial amount of distance away from the guard.

Test Status: Pass

<u>ID# - FT15</u>

Date(s) of Execution: 11/25/2020

Staff conducting tests: John Mistica

Expected Results: Multiple guards are able to follow the player at once

Actual Results: Multiple guards were successfully able to follow the player at once

Test Status: Pass

ID# - FT16

Date(s) of Execution: 11/27/2020

Staff conducting tests: Julian Gonzales

Expected Results: Player will move in all directions including diagonal

Actual Results: Player moves in every direction

Test Status: Pass

ID# - FT17

Date(s) of Execution: 11/27/2020

Staff conducting tests: Julian Gonzales

Expected Results: Player will collide with a wall and stop

Actual Results: Player immediately stops after colliding with wall

Test Status: Pass

ID# - FT18

Date(s) of Execution: 11/27/2020

Staff conducting tests: Julian Gonzales

Expected Results: Player will collide with npc proceed to fail

Actual Results: Player fails immediately after colliding

Test Status: Pass

ID# - FT19

Date(s) of Execution: 11/27/2020

Staff conducting tests: Julian Gonzales

Expected Results: Player will stop moving after pausing the game

Actual Results: Movement for the player is prohibited and npc also stop

Test Status: Pass

11 Regression Testing

Since the above tests were the first tests created for this project, there was no regression testing done for the project.

IV Inspection

12 Items to be Inspected

Items to be inspected in this report are the same as listed in Section 8.

13 Inspection Procedures

For inspecting our code we repurposed a Java code checklist from the University of Toronto [2]. Since our code is primarily written in C#, there are few differences for the purposes of inspection.

For each piece of code submitted for inspection, the three group members not responsible for the code met electronically over Discord during a single 15 minute meeting and went through the checklist. In addition the inspectors also looked through the code and noted personal notes about the structure, style and any possible improvements to the code.

14 Inspection Results

Player Movement: MoveCharacter.cs

Inspectors: Julian Gonzales, Dan Hrubec, John Mistica

Time of Inspection: 4:45pm 11/27

Results: Passing

All sections of the code pass with the exception of section 10 in reference to comments within the code. The code has little to no comments explaining what each piece of code is for. A run through of comments explaining what each function's preconditions and postconditions are as well as what each of the class variables are used for are needed to improve this file. All other aspects of the code follow the guidelines listed in the inspection checklist.

The variable naming is descriptive and gives insight as to what it is being used for, all conditions are checked and accounted for, as well as all arguments passed as parameters are being used and checked beforehand. The file is clean and contains little to no repetitive or redundant code.

In conclusion, the main issue with the file is the lack of comments to describe to others its usage. All other aspects of the code pass and maintain proper functionality, so the inspection of this file passes.

Map Storage/Loading: GridScript.cs

Inspectors: Julian Gonzales, Jake DiStefano, John Mistica

Time of Inspection: 3:30pm 11/27

Results: Passing

It passed all sections except section 1, section 8 and section 10. For section 1, some of the names where confusing and very similar. Some of the names had un distinguishable names like tile1 and tile2. More descriptive names could have been used for these cases to distinguish the difference between the two and make it easier to understand. All the methods were named with descriptive names that gave a description of what the methods do.

For section 8 the reading of the csv file was not closed which does not properly dispose of unneeded resources. Other then the csv file no other file was needed but multiple levels are loaded throughout the game which could add up for large csv files.

Section 10 some comments were given but some methods do not have comments explaining their functionality or only one at the start of the method. An explanation of each method should be done to improve the readability.

The inspection team approves this segment of code since it only fails minor tests involving readability and commenting. The test failed in section 8 can easily be fixed and does not involve much work.

Item and Inventory: Item.cs, ItemFactory.cs, Inventory.cs

Inspectors: Jacob DiStefano, Dan Hrubec, John Mistica

Time of Inspection: 5:15pm 11/27

Results: Passing

The style of the code is passing, following standard camelcase expectations. However one filename does not quite match its classname. In reference to Section 10 of the checklist, all files have a severe lack of comments. Beyond this however no defects were found in the code that could be seen. Code style is generally clean and consistent throughout these files.

The use of patterns throughout the files looked solid and consistent. The inspectors appreciate the use of the factory design pattern for items, and the inner class ItemObject for use in storing objects. However the inspectors did note that the constructor for InventoryObject does seem to be doing a little too much work, and some of its functionality should be moved into a private method outside the constructor, for clarity. Another thing that concerned the inspection team was the hardcoding of the amount of items in ItemFactory.cs, generateRandomInt(). A "amount of items member" exists, yet its value is never referenced.

Over all the inspection team approves of this segment of code, it failed some minor stylistic checks on the checklist, but beyond that it is respectable code.

Guard Pathin/AI: GuardPathing.cs

Inspectors: Jacob DiStefano, Julian Gonzales, Dan Hrubec

Time of Inspection: 9:30pm 11/27

Results: Passing

The code passes most sections of inspection, however there are a few minor adjustments that could be made. In reference to section 1 (Variable, Attribute, and Constant declaration Defects), there are some variables that should have there access modifiers adjusted appropriately such as some local variable names to be made private. This is a minor fix which doesn't affect the overall functionality of the code.

In regards to section 10 (Comment Defects), most of the code passes, however there could be minor adjustments made that would add to the readability of the code. Some variables were left uncommented which would help aid in understanding of the code. Overall, more comments should be added, but there is enough to understand the purpose and function of the code.

The layout and style of the code is streamlined and efficient. There are some minor defects such as indentation errors, but most methods and compile modules are short and to the point. In reference to section 11 (Layout and Packaging Defects), indentation stays pretty consistent throughout the entire file, besides the previously mentioned errors.

Overall, the style of the code was streamlined and minimal. The AI developed was obviously minimal and could be made more dynamic, however the code allows for further extension and modularity. The central consequence of the lack of comments/indentation errors was readability, which can be quickly fixed by making a few minor adjustments. The code upholds its intended functionality with the exception of a few minor stylistic errors, therefore the inspection team approves of this code

V Recommendations and Conclusions

Player Movement: MoveCharacter.cs

Testing: Passed

Inspection: Passed

Future Actions: Additional comments to code

Map Storage/Loading: GridScript.cs

Testing: Passed

Inspection: Passed

Future Actions: Optimize code to be able to handle larger files and refactor code with some stylistic adjustments such as additional comments.

Item and Inventory: Item.cs, ItemFactory.cs, Inventory.cs

Testing: Passed

Inspection: Passed

Future Actions: Additional stylistic adjustments such as comments.

Guard Pathin/AI: GuardPathing.cs

Testing: Passed

Inspection: Passed

Future Actions: Develop AI to be more dynamic and implement a few minor stylistic elements.

VI Project Issues

15 Open Issues

OI#1 - Map Generation:

Content: We would like to implement a more streamlined and efficient way to generate the map, but have not yet made any significant changes/conclusions.

Motivation: Generating larger maps require a large amount computation power, therefore crashing machines when some large maps are generated.

OI#2 - Dynamic AI:

Content: We would like to implement a more dynamic AI w/ more sophisticated pathfinding algorithms, but have not yet come to research an optimal solution.

Motivation: Having smarter AI allows for players to have a more challenging experience and creates a more polished gameplay experience.

OI#3 - Profit:

Content: Our application currently has not implemented any model to generate profits and we have not yet researched any ways to implement this.

Motivation: Future development of this application may require developmental costs that potential stakeholders would provide.

16 Waiting Room

WR#1 - Timer:

Content: In a future release we wish to implement a timer into each level of the game.

Motivation: Implementing a timer would add more depth and interest to the game which would be more enjoyable for the player. A timer would also fit the theme of our "Prison Break" game and add further immersion.

WR#2 - Additional Item Functionality:

Content: In a future release we wish to implement more item functionality and variety.

Motivation: Implementing more item functionality would allow players to have a better gameplay experience by having more ways to beat the game at their disposal.

<u>WR#3 - Animations:</u>

Content: In a future release we wish to implement more player and item animations.

Motivation: Having animated sprites allows for a more polished and streamlined look which would provide a much more pleasurable gameplay experience for the player.

<u>WR#4 - Lighting:</u>

Content: In a future release we wish to implement dynamic lighting.

Motivation: Implementing dynamic lighting would allow for not only a more polished look to the game, but also more depth and challenge which in turn allows the player a better gameplay experience.

17 Ideas for Solutions

<u>IS#1 - Map Generation:</u>

Content: Currently using 2D arrays that represent all tiles on the map which is inefficient when representing empty tiles. These could represented without the use of memory in a more complex data structure

Motivation: This would decrease the use of memory significantly and possibly stop the current issue of crashing with large maps

<u>IS#2 - Dynamic AI:</u>

Content: A Dynamic AI could be implemented using the A* pathfinding package which easily creates the pathfinding AI on its own or implement the feature from the ground up using the tiles to do a shortest path algorithm to the players position

Motivation: This would add complexity to the game and add options for the AI, one could be more challenging over the other

<u>IS#3 - In App Purchases:</u>

Content: In app purchases could be implemented creating different skins for the player, different maps that can be unlocked. A monetary system could be added to the game which allows the player to acquire coins for each game they win and then those could be used to purchase skins. These coins can also then be purchased with real money.

Motivation: This would create profits that users can also enjoy and not distract them from the game unlike other forms of profits like ads

<u>IS#4 - Advertisements:</u>

Content: Advertisements can be shown every few rounds of games making sure they are not bothering the user excessively.

Motivation: Would create a steady profit but also negatively impact the user experience so use with caution.

18 Project Retrospective

PR#1 - CSV Map Creation:

Content: Map Creation from csv file worked really well and made it easy to create new levels. The CSV files could also be auto generated in the future.

Motivation: This removes the repetitive process of dragging and dropping stuff in unity which could add bugs to the game and instead just create the map using a script and file

PR#2 - Item Factory:

Content: Items generated from a factory can easily add other forms of generating items. Was very scalable when radom generation was needed and wasn't interfered by new items being added.

Motivation: Factories could be used to implement other types of generating like enemies and npc's

PR#3 - AI:

Content: The AI could have been done better since they would at times get stuck with walls and stay within the same general areas. Maybe creating a way to avoid walls and randomly walk towards open areas.

Motivation: It would add difficulty to the game since players could avoid the path of the enemy instead of just avoiding the general area

PR#3 - Player Movement:

Content: Was able to move in all directions including diagonally which was helpful in avoiding enemies

Motivation: Doing so avoids the controls being an obstacle of the game and the user can focus more on the game.

VII References / Bibliography

[1] Saikrishna Yadavalli, Hassam Hussain, Tommy Castino, Joshua Peterson, Prison Break Project Report

[2] University of Toronto, Java Inspection Checklist https://www.cs.toronto.edu/~sme/CSC444F/handouts/java_checklist.pdf