**Prison Break Final Report Summary**

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

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. 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.

This project was initially proposed by Group 9 from the Spring 2019 semester. 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.

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. The first release was primarily dedicated to infrastructure setup. This included the infrastructure for map creation, storage and loading via csv files. It also included the user interface and basic controls for the game, without any specialized art. The second release allowed us to add the primary functionality of the project, and to polish what was already there with smoother gameplay and more fitting art. The work on this release included guard AI, item functionality, level progression and failure/victory states. This release was also polished by adding more fitting art and fixing bugs.

For this report we chose to test four elements of functionality for the project, spanning multiple files. We verified the functional requirements for Items and Inventory, Map Storage and Loading, Guard AI, and Player Movement. These elements were tested manually by each team member, by creating a custom map or element and verifying the behavior of a specific piece of functionality in that case.

During Dan Hrubec’s testing of the Items and Inventory functionality, some errors were found. While the primary functionality of the inventory was found to be functional, faults were detected when the user has a full inventory. When a user has a full inventory, they should not be able to pick up more items and it should not be removed from the map, and yet it was.

During Jake DiStefano’s testing of the Map Storage and Loading functionality, some more errors were found. The utility that loads maps into Unity should be able to reject files that are incorrectly formatted, yet it will attempt to load the map without any error. Another error discovered was the support of large maps, which were not able to load, and actually crashed his computer. Beyond this the normal functionality of Map Loading and Storage was successful.

During John Mistica’s testing of Guard AI, only one error was found. One of the critical functionalities of Guard AI is that they should stop searching for the player, however this does not happen currently. Beyond this the Guard AI is functional.

For inspecting our code we repurposed a Java code checklist from the University of Toronto. 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.

Every team member but the author Jacob DiStefano inspected the files relating to character movement and collisions. The inspectors went through the checklist and inspected the code style and design. 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.

Every team member but the author Dan Hrubec inspected the files relating to Map Storage and Loading. After going through the checklist, 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 indistinguishable 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.

Every team member but the author Julian Gonzales inspected the files relating to Items and Inventory. 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. Overall this piece of code was very solid, if not a little uncommented.

Every team member but the author John Mistica inspected the files relating to Guard Pathing and AI. 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.