Prison Break [1]

Group 19

Mitchell Jones, Dylan Brunelle, Dua'a Hussein, Daniel Kim

CS 440

at the

University of Illinois Chicago

November 2023

Table of Contents

List of Figures 3

List of Tables 3

I Project Description 4

1 Project Overview 4

2 Project Domain 4

3 Relationship to Other Documents 4

4 Naming Conventions and Definitions 4

4a Definitions of Key Terms 4

4b UML and Other Notation Used in This Document 5

4c Data Dictionary for Any Included Models 6

II Project Deliverables 6

5 First Release 7

6 Second Release 8

7 Third Release 10

8 Comparison with Original Project Design Document 13

III Testing 14

9 Items to be Tested 14

10 Test Specifications 14

11 Test Results 24

12 Regression Testing 30

IV Inspection 30

13 Items to be Inspected 30

14 Inspection Procedures 30

15 Inspection Results 31

V Recommendations and Conclusions 32

VI Project Issues 33

16 Open Issues 33

17 Waiting Room 33

18 Ideas for Solutions 34

19 Project Retrospective 34

VII Glossary 35

VIII References / Bibliography 36

IX Index 37

List of Figures

Figure 1 - Sample UML Diagram 6

Figure 2 - First Release In-Game Screenshot 7

Figure 3 - First Release UML Diagram 8

Figure 4 - Second Release Updated Sprites 9

Figure 5 - Second Release Expanded Map 9

Figure 6 - Second Release Item Variety 9

Figure 7 - Second Release Furnished Room 9

Figure 8 - Second Release UML Diagram 10

Figure 9 - Third Release Item Demonstration 11

Figure 10 - Third Release Player Disguise 11

Figure 11 - Third Release Escape Room 12

Figure 12 - Third Release Win Screen 12

Figure 13 - Third Release UML Diagram 13

List of Tables

N/A

I Project Description

1 Project Overview

Prison Break is a puzzle game where the player is a prisoner who is serving their sentence at the local prison. During a total blackout that affects the entirety of the facility, the player discovers that their cell is unlocked and that they are able to leave their cell. With an opportunity to escape that does not come often, the gameplay involves the player maneuvering around the prison, collecting useful items, solving puzzles, and avoiding guards and obstacles until they find a means of escaping the prison and achieving freedom.

2 Project Domain

There is no particular material being tested with the creation of the Prison Break game that needs its value or correctness evaluated accordingly.

3 Relationship to Other Documents

One of the main components used for the development of Prison Break was the Project Final Development Report created by the team prior to the current team. Members of the prior development team who produced the development report were Saikrishna Yadavalli, Hassam Hussein, Tommy Castino and Joshua Peterson [1]. The current development of Prison Break utilized the final development report to create a basis for its most basic functionality and requirements, as well as a guide of sorts to how the current team should proceed with creating and implementing certain features.

The two scenario documents created by the team showcase the broader development timeline of the game, starting with moving a character around a room, to creating a larger map of rooms linked together, populated by obstacles, items and guards.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Entity - Any physical object in the game that can be placed in a room and possess characteristics, such as the player, items, obstacles, decorative objects, room-tiles, collisions-tiles, and guards.

Component - Any characteristic that can be attached to an entity by means of a .h file extending the Component class. These would include an entity's transform component (location/size of entity), sprite component (visual representation of the entity), collision component (the ability to collide with other entities who possess this component), etc. Entities can have the same components as other entities but each instance of a component is unique to that particular entity.

Object - A generic characteristic component of an entity having both a transform and sprite component.

Item - A component of an entity that allows it to be picked up by the player as well as possess a unique id

Obstacle - A component of an entity that makes the entity restrict the player's movement and be removed from the room by the player if he/she is holding an item with an id matching the obstacle's key variable.

NPC - A component that allows an entity to follow a path of nodes supplied to it. In Prison Break, the main NPCs are prison guards.

Room - An instance of the Room class which possesses a file path to tilemap, containing both the locations of tiles to be rendered to the screen as well as information like the location of wall "colliders" and Navmesh nodes. It also possesses a list of objects to be loaded upon the player entering the room, and two dimensional vectors containing the locations of each entrance/exit of the room, one for each cardinal direction.

Map - An array of rooms which make up the prison as a whole. The map allows the exiting of one room to return an instance of another, so that the player can make their way throughout the prison.

Navmesh - The "Navigation Mesh" method of pathfinding used to supply the NPCs with paths to move throughout their rooms. Prison Break uses a set of nodes generated from room text files and Dijkstra's shortest path algorithm in its Navmesh class in order to create smooth AI movement. While not a true "Navmesh" as seen in popular game engines like Unreal or Unity, the navmeshes seen in these engines served as inspiration for AI navigation as seen in Prison Break.

4b UML and Other Notation Used in This Document

The UML diagrams present in this document follow the symbol definitions as seen in *UML Distilled* by Fowler. Below is a reference for these symbols.

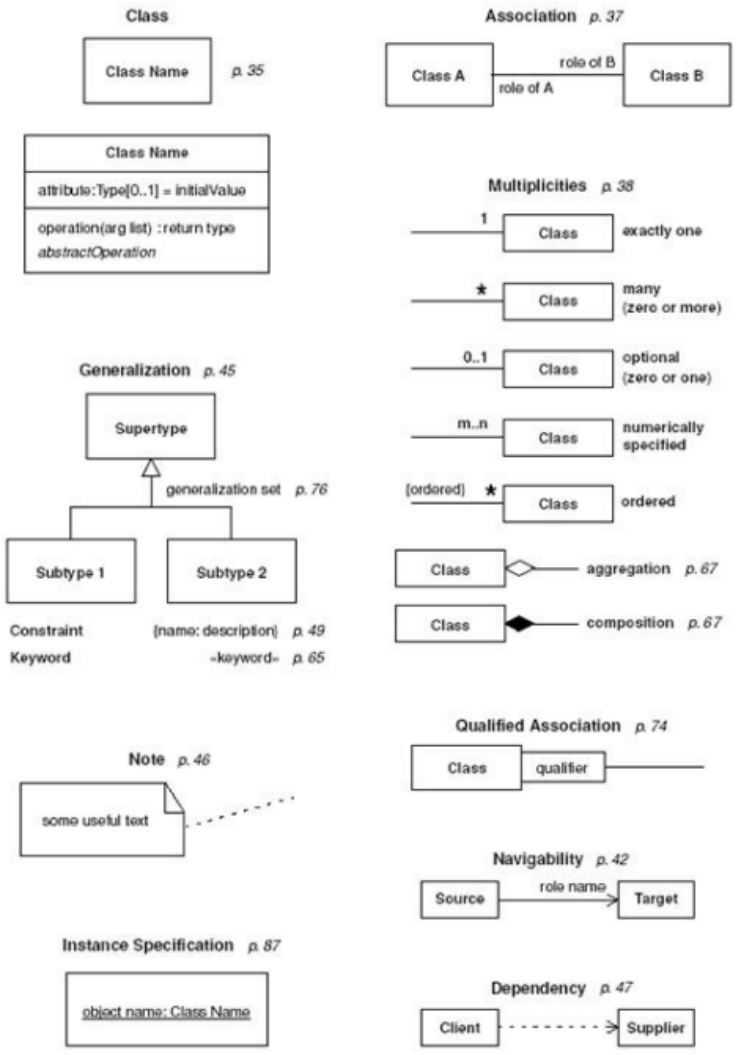


Figure 1 - A sample UML diagram that will be used throughout this document

4c Data Dictionary for Any Included Models

Asset file types:

Tilesets (image containing a collage of every background tile in the game) → .png

Tilemaps (data representation of a room in the form of an array of integers) → .txt

Sprite-sheets (a set of images in a single file used to create an animation) → .png

Objects, items, obstacles (unique physical objects found in the game) → .png/.jpg

II Project Deliverables

Over the course of the semester, the team produced a working version of Prison Break that allows the user to roam around the map of the prison, find useful items, interact with the environment, and achieve an end goal of escaping the prison.

5 First Release

The first release of Prison Break occurred on October 6th, 2023, with basic functionality. The player was able to navigate around a small room with a hallway that is blocked off by a fire. The walls seen present in the level as well as the fire possess a collision trait, preventing the player from passing through them.

There is a fire extinguisher that is in the corner of the map, but as of that time, the player was unable to use it. The sprites used for the prisoner sprite and fire were placeholders that were static images as they did not have animations.

However, the focus for this release based off of the scenario written beforehand was room generation from a text file and basic player movement, which the team successfully implemented and demonstrated.

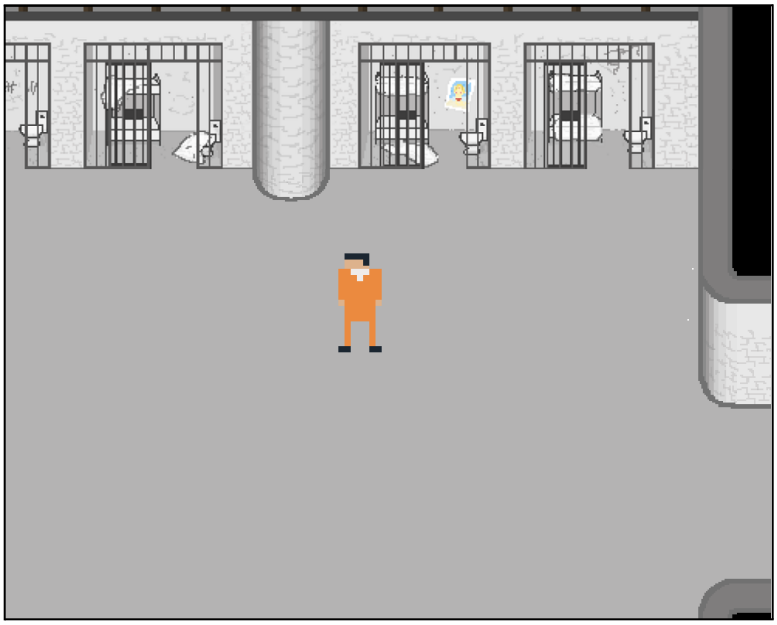


Figure 2 - Documentation of the first release of Prison Break, featuring the original sprite used for the player

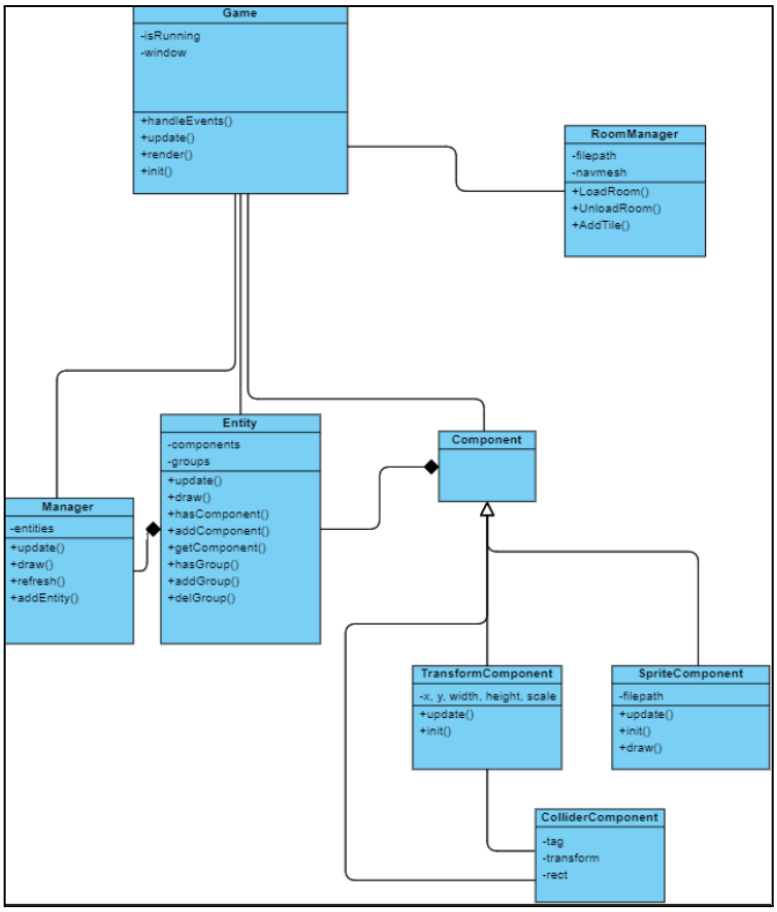


Figure 3 - UML Diagram demonstrating the functionality of the first release of Prison Break

6 Second Release

The second release of Prison Break occurred on November 4th, 2023. Player navigation received a massive upgrade, allowing the player to traverse an entire map of several different rooms, linked together via a MapManager class.

The player now has several options of items that they can find and use to overcome the many new obstacles added with the second release, although the items at this point are all in the same area for debugging purposes and not dispersed throughout the map yet.

Upon use on an obstacle, the item the player is holding disappears and the player is not able to use it again. While no definite escape has been implemented at that point in time, the team achieved the goal of cross-room navigation as well as implementing item-obstacle interaction, the goal of the second release as well as the key components of Prison Break gameplay.

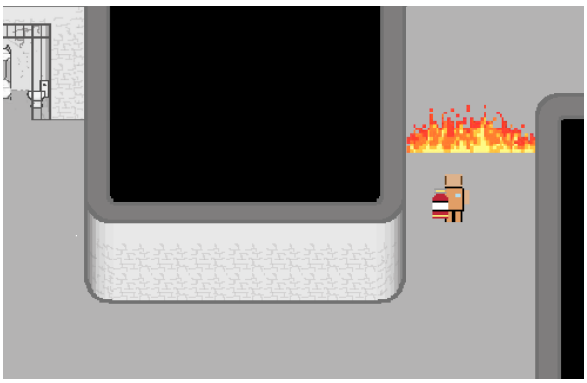
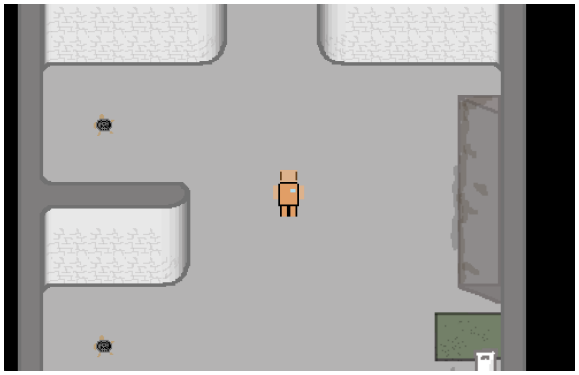
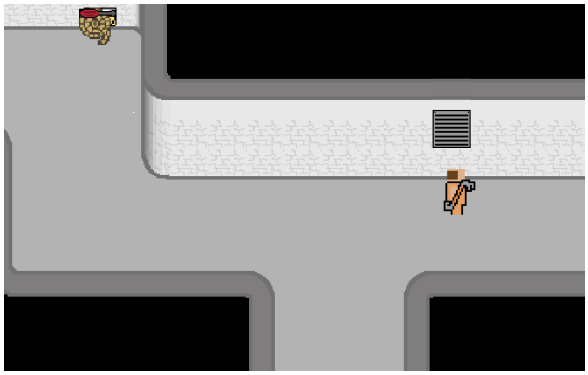


Figure 4 - In game screenshot of the second release, with updated sprites





Figures 5 & 6 - The player is able to walk around the prison and hold different items

Figure 7 - Player in a furnished room as of the second release of Prison Break

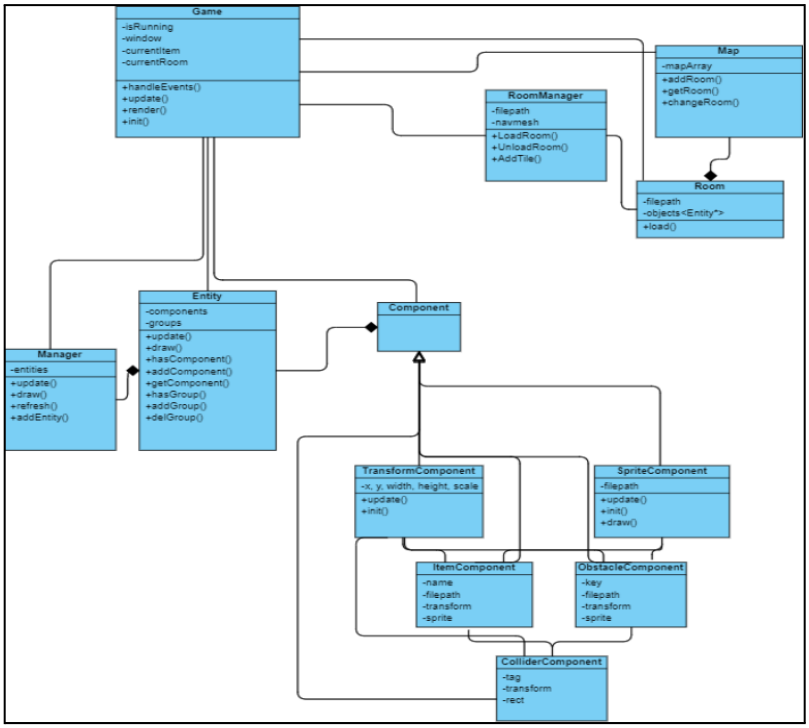


Figure 8 - UML Diagram of functionality and code organization as of the second release of Prison Break

7 Third Release (Presentation)

The third and most recent release of Prison Break was dropped on November 21st, 2023. The final items and obstacles were added, and the player, through the clever use of pairing particular items to their obstacles, can escape the prison, winning the game.

Most importantly, however, was the introduction of the Guard mechanic to the game. Now several guards populate the map, patrolling rooms using a pathfinding algorithm. If the player is spotted, the guards will rapidly move towards the player, and if enough distance is closed, will shoot the player as well, allowing the player to now lose the game.

Several other non-obstacle related items were added in order to give the player an edge over the guards, including an energy drink that increases the player's speed, a knife that can be used to attack the guards when they aren't looking, and a guard disguise that will camouflage the player.

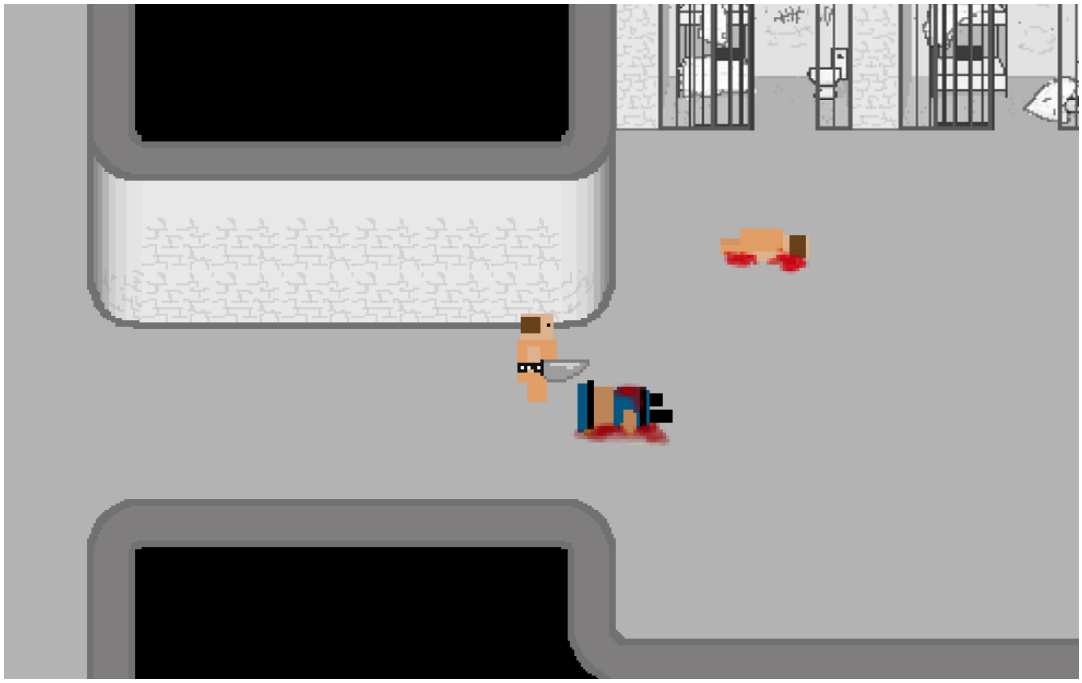


Figure 9 - The player demonstrating the use of items on NPCs in game

Figure 10 - The player disguised in a guard outfit to deceive the guard NPCs

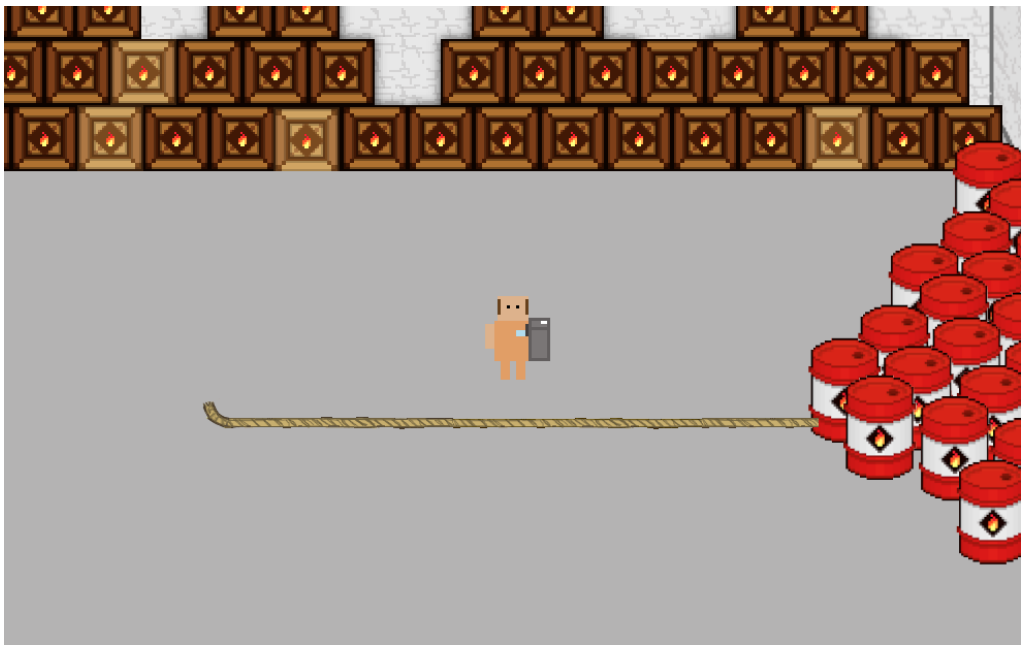


Figure 11 - The player holding a lighter in the ammunitions room

Figure 12 - The player escaping the prison through a hole created by an explosion

caused by the ammunition room exploding

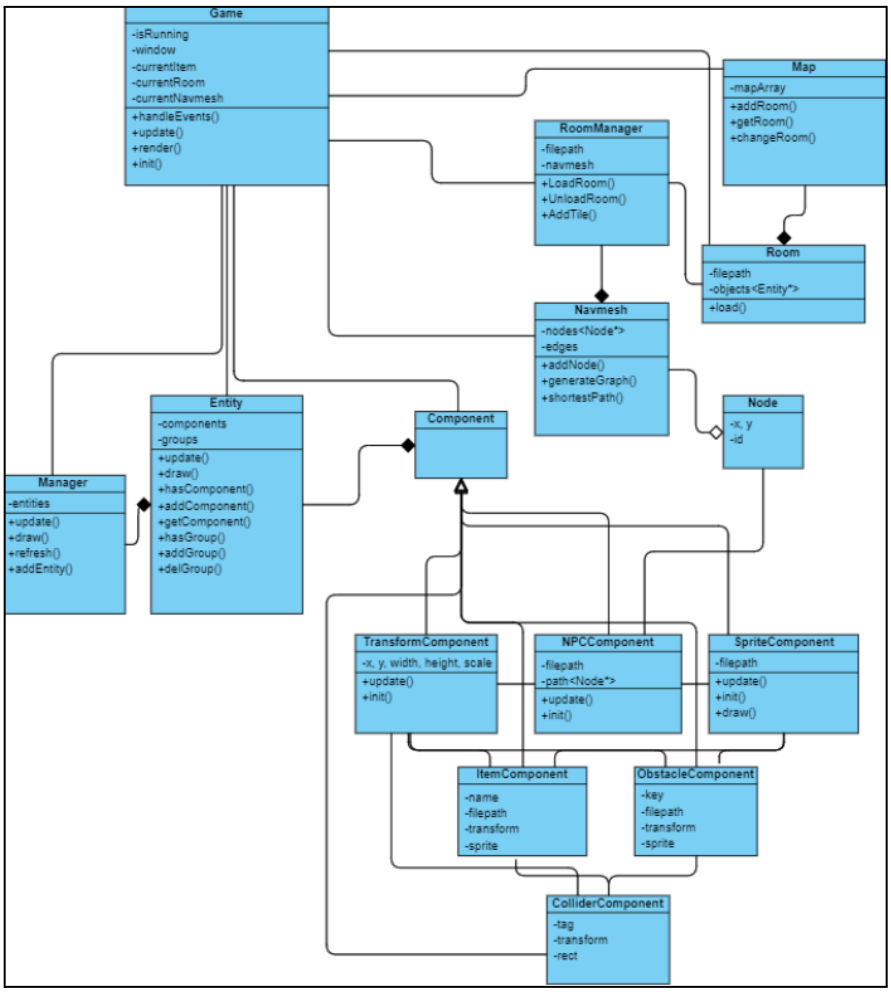


Figure 13 - A UML diagram that demonstrates the updated functionality and code structure of the third release of Prison Break

*Note that the children of the Component class shown are only some examples, more components can be applied to Entities in the current state of the game, such as a keyboard controller, or background tile component.

8 Comparison with Original Project Design Document

In its most current version, Prison Break contains much of the basic functionality outlined in the original project design. The premise of the prisoner escaping their cell during a total power outage and using objects they find to aid their escape is demonstrated, as described by the tutorial, is 100% present in the actual gameplay. The items, obstacles, and scenarios that are present in the current version are outlined as possible events that can occur according to the original design plan.

In contrast to the original design document, the current version of Prison Break does not support multiplayer that was intended in the original design and only supports

single player at the moment. Alongside that, Prison Break as it currently is only has one static map, instead of the constantly randomized map that was noted in the design. However, with modular design of the rooms, as well as an entity-component system that allows for the creation of new entities (such as fellow prisoners), the current state of the project would allow for the seamless addition of these features, without changing the core mechanics present.

III Testing

9 Items to be Tested

This project does not lend itself to unit testing due to the interdependence of classes and components. Instead, functionality testing was mainly utilized. Functions, once implemented, were tested by playtesting the game to verify that the intended functionality the function was supposed to provide was present in the game.

Functionalities that were tested:

- Start Screen navigation
- Room-generation
- Map-generation
- Room-switching
- Entity components
- Collision detection
- Displaying sprites
- Sprite-switching
- Sprite animations
- Player/Item/Obstacle/Decoration spawning
- Player movement
- Adding items/obstacles/decorations to room (required Player/Item/Obstacle/ Decoration spawning)
- Picking up items
- Item-switching
- Interactions between items and obstacles
- Guard AI pathfinding
- Lose game sequence
- Win game sequence

10 Test Specifications

#00 - Start Screen Navigation

Description: Game start screen with a start button to play the game, a how-to-play button for instructions, and a quit button to quit the game.

Items covered by this test: StartScreen.h, StartScreenIcon.h

Requirements addressed by this test: Requirement #23

Environmental needs: N/A

Intercase Dependencies: Test 02. In order to test the "start game" button, there must be some aspect of the main game to start (in this case, room generation).

Test Procedures: Run the game. Check for a start screen with a start button, how-to-play button, and a quit button. Make sure clicking the start game button starts the main game. Make sure the how-to-play button displays a list of instructions and controls. Make sure the quit button closes the game window.

Input Specification: Button clicks, one for each button.

Output Specifications: Clicking the start game button starts the main game. The how-to-play button displays a list of instructions and controls. The quit button closes the game window.

Pass/Fail Criteria: Clicking the start game button starts the main game. The how-to-play button displays a list of instructions and controls. The quit button closes the game window. If all of these are true, the test passes. If not, the test fails.

#01 - Entities and Components

Description: Entities can have components added to them, adding characteristics to them.

Items covered by this test: Manager, Entity, and Component classes,

Requirements addressed by this test: N/A.

Environmental needs: N/A.

Intercase Dependencies: N/A.

Test Procedures: Create an entity, and the transform entity to it. Print the entity's position to console. Set the entity's position to something else. Print the position to the console again.

Input Specification: The positions of the entity.

Output Specifications: The values displayed to the console..

Pass/Fail Criteria: If the console displays the old and new positions of the entity, the test succeeds. If not, the test fails, and the component may not have been properly attached to the entity.

#02 - Game sprites

Description: Entities are properly visually represented by sprites.

Items covered by this test: Displaying sprites, Sprite-switching, Sprite animations

Requirements addressed by this test: N/A.

Environmental needs: N/A.

Intercase Dependencies: Test 01, entities must be able to have a sprite component added to them.

Test Procedures: Create an entity, and add a sprite component to it. Create another entity but use an animated sprite.

Input Specification: File path of sprite .pngs, and the entities' transform components.

Output Specifications: The entities rendered to the game window.

Pass/Fail Criteria: If the correct sprites are rendered to their correct positions with correct animations if enabled, test succeeds, if not test fails.

#03 - Room-Generation

Description: Room data loaded from a text file, is rendered to the game display.

Items covered by this test: Room .txt files, Room class, RoomManager class

Requirements addressed by this test: Requirements 02, 03, 04. Room is generated in a precise, timely fashion.

Environmental needs: N/A

Intercase Dependencies: Required for Test 04. Requires Test 01 to add collider entities with collider components to the room.

Test Procedures: Start a new game, with currentRoom set to a new Room instance constructed with the room's dimensions and the room text file's filepath.

Input Specification: The correct text file file-path associated with the desired level.

Output Specifications: The current level (room) should be loaded with no issues, and properly rendered to the game's display window.

Pass/Fail Criteria: Room incorrectly rendered, i.e. tiles in incorrect locations, or

colliders in spots causing the player to seemingly collide with nothing would result in a test failure. Room correctly rendered, with the player only colliding with desired walls is a test success.

#04 - Room-Map Management

Description: A 2D array of Rooms is generated allowing the exit of one room to result in the loading/display of the neighboring room, depending on the direction the player is traveling from.

Items covered by this test: Map-Generation (MapManager.h), Room-Switching

Requirements addressed by this test: Requirement F1. Note: While the current release of Prison Break does not have random generation implemented, the MapManager class was designed to support it in the future. Requirements 02, 03, 04.

Environmental needs: N/A

Intercase Dependencies: Requires a successful Test 03 in order to generate and properly render new rooms after the player leaves the current one.

Test Procedures: Set up the 2D array of Rooms using the MapManagers addRoom() function. Initialize the currentRoom variable with one of the rooms in the matrix. Start the game, and move the player outside the room via one of the hallways. The current room should be unloaded and the desired room should be loaded in its place with the player being moved to the desired location. This test should be repeated in all directions in multiple rooms

Input Specification: A properly constructed 2D array of rooms.

Output Specifications: The rooms generated as the player traverses the map.

Pass/Fail Criteria: If the player leaves the currentRoom and ends up in a different room than desired, the test fails. If the player leaves the currentRoom and ends up in the correct room, but in the wrong location, the test fails. If the player leaves the currentRoom and ends up in the correct room and location, the test succeeds. This test must succeed in all directions (north, east, south and west).

#05 - Colliders

Description: Entities may possess colliders, allowing the game to check if two entities are overlapping.

Items covered by this test: Colliders

Requirements addressed by this test: N/A.

Environmental needs: N/A.

Intercase Dependencies: Test 01, entities must be able to have a collider component attached to them, Test 03, the room must have colliders to check if the player is colliding against them.

Test Procedures: Create a player entity, and add a collider component to it. Move the player into a wall where there is a collider entity. In the update() loop, check if the player's collider overlaps a wall's collider using the Collision::AABB(collider A, collider B) function. If true, stop player movement.

Input Specification: Player movement

Output Specifications: The player entities resistance to movement

Pass/Fail Criteria: If the player stops moving when reaching the wall, success. If the player passes through, fail.

#06 - Guard Pathfinding

Description: Guards correctly follow the shortest path of nodes from their current location to a destination location.

Items covered by this test: Navmesh class, NPC Component

Requirements addressed by this test: Requirements 03, 04. Guard pathfinding does not affect performance of the game.

Environmental needs: N/A

Intercase Dependencies: Requires a successful Test 03 to ensure Navmesh contains proper nodes.

Test Procedures: Set currentRoom to a room containing Navmesh data. Create a guard entity (possessing an NPCComponent) and add it to the room (room.addObject()). Start the game.

Input Specification: A vector of nodes generated by getShortestPath() from src, the guards closest node, and dest, the player's closest node.

Output Specifications: The path the guard visually takes.

Pass/Fail Criteria: If the guard follows the player, test success. If not, the test fails.

#07 - Guards Kill Player

Description: Guards kill players if the player gets too close while the guard faces them

Items covered by this test: Lose Game sequence.

Requirements addressed by this test: Requirement F3.

Environmental needs: N/A

Intercase Dependencies: Requires a successful Test 10 to ensure a guard can properly navigate to the player.

Test Procedures: Set currentRoom to a room containing Navmesh data. Create a guard entity (possessing an NPCComponent) and add it to the room (room.addObject()). Start the game, and stand in front of the guard. Wait for the guard to approach.

Input Specification: Player movement.

Output Specifications: Whether the guard pulls out a gun or not.

Pass/Fail Criteria: If the guard kills the player, while they are right in front of them, test success. If not, the test fails.

#08 - Entity Spawning

Description: Entities (items/obstacles/decorations) are placed/spawned into specified rooms at a specified location, with the size, sprite, and collider properties dictated by the components attached to those entities.

Items covered by this test: Entity Room Placement, Room class, ItemComponent.h, ObstacleComponent.h, ObjectComponent.h

Requirements addressed by this test: Requirement 04. The game uses stored data within Component classes attached to entities to determine the correct size/sprites/colliders to give the entity when spawning it into the room.

Environmental needs: N/A

Intercase Dependencies: Tests 01, 02, 03. To test entity spawning, entities must first exist, sprites must be given to them so that they are visible/recognizable, and rooms must exist to spawn them in. Tests 10 and 12 depend on test 08.

Test Procedures: Create an entity that has an ItemComponent, ObstacleComponent, or ObjectComponent (which, in turn, gives the entities a TransformComponent, SpriteComponent, and/or ColliderComponent). Create a room that the entity should be placed within. Spawn the entity into the room by calling the room's AddObject() function, passing in as arguments a reference to the entity and the x and y position the entity should be placed within the room, being mindful of the room shape/bounds. Start the game, making sure the starting room is set to the room the entity was spawned in for testing purposes. Navigate to

the (x,y) position of the room that was passed as parameters to the AddObject() function.

Input Specification: A reference to the entity to add to the room, the x position to place the entity within the room, the y position to place the entity within the room. After the game is started, player movement using the 'W'/'A'/'S'/'D' keys to navigate to the location in the room the entity should have been placed in.

Output Specifications: It is expected that upon navigating to the (x,y) position of the room specified in the AddObject() function when spawning the entity to the room, the entity will appear with all the appropriate properties specified by its components.

Pass/Fail Criteria: The test passes if the entity appears in the correct room in the correct location, with the correct properties given to it through ItemComponent, ObstacleComponent, or ObjectComponent. Otherwise, the test fails.

#09 - Player Movement

Description: The player is able to move in all four cardinal directions (north, east, south, west), or a combination of north/south and east/west.

Items covered by this test: Sprite Switching, KeyboardController.h

Requirements addressed by this test: Requirement 02. There is very little (imperceivable) latency between 'W'/'A'/'S'/'D' keypresses and the player moving in the appropriate direction.

Environmental needs: N/A

Intercase Dependencies: Tests 01, 02, 03. To test player movement, an entity must be created for the player with the appropriate player sprite and there must be a room to move within.

Test Procedures: Start the game. Press the 'W' key and release it immediately. Press the 'D' key and release it immediately. Press the 'S' key and release it immediately. Press the 'A' key and release it immediately. Repeat this process, but this time holding the keys for a few seconds. Press these combinations of keys: 'W' and 'A', 'W' and 'D', 'S' and 'A', 'S' and 'D'.

Input Specification: 'W'/'A'/'S'/'D' keyboard presses.

Output Specifications: When 'W' is pressed but not held down, the player should move up (north) only slightly and then stop. When 'A' is pressed but not held down, the player should move left (west) only slightly and then stop. When 'S' is pressed but not held down, the player should move down (south) only slightly and then stop. When 'D' is pressed but not held down, the player should move right (east) only slightly and then stop. When 'W' is pressed and held down for a few

seconds, the player should move up (north) for the duration of the keypress and then stop. When ‘A’ is pressed and held down for a few seconds, the player should move left (west) for the duration of the keypress and then stop. When ‘S’ is pressed and held down for a few seconds, the player should move down (south) for the duration of the keypress and then stop. When ‘D’ is pressed and held down for a few seconds, the player should move left (east) for the duration of the keypress and then stop. When ‘W’ and ‘A’ are pressed simultaneously, the player should move northwest. When ‘W’ and ‘D’ are pressed simultaneously, the player should move northeast. When ‘S’ and ‘A’ are pressed simultaneously, the player should move southwest. When ‘S’ and ‘D’ are pressed simultaneously, the player should move southeast.

Pass/Fail Criteria: If the expected behavior laid out in “Output Specifications” is observed, the test passes. Otherwise, the test fails.

#10 - Item Pickup

Description: When a player is standing on top of an item and the ‘E’ key is pressed, the player will pick up the item, which is added to the player’s inventory.

Items covered by this test: Sprite Switching, Entity Spawning, KeyboardController.h

Requirements addressed by this test: Requirement 02. Picking up multiple items does not cause the game to lag.

Environmental needs: N/A

Intercase Dependencies: Test 08. To test item pickup, items must be able to be spawned within rooms. Tests 11 and 12 depend on test 10.

Test Procedures: Start the game, making sure to set the starting room as one that has an item placed in it and was tested in test 08. Move the player so that he is standing on top of the item. Press the ‘E’ key on the keyboard.

Input Specification: ‘W’/’A’/’S’/’D’/’E’ keyboard presses.

Output Specifications: When ‘E’ is pressed when the player is on top of the item, the item should disappear and the player sprite should change to one where the player is holding the item that was just picked up.

Pass/Fail Criteria: If the expected behavior laid out in “Output Specifications” is observed, the test passes. Otherwise, the test fails.

#11 - Item Switching

Description: After a player has picked up multiple items, the player can cycle/scroll through the items they have in their inventory using the ‘Q’ key.

Items covered by this test: Sprite Switching

Requirements addressed by this test: Requirement 02. Switching between items in the inventory is seamless, ensuring the player is able to switch to a certain item quickly with little trouble.

Environmental needs: N/A

Intercase Dependencies: Tests 04, 10. To test item switching, the player first must be able to pick up multiple items. In case those items are located in different rooms, the player must be able to navigate to/from rooms. Test 12 depends on test 11.

Test Procedures: Create two items and place them in different rooms. Start the game. Navigate to a room containing one of the items. Press 'E' to pick it up. Then navigate the room with the other item. Press 'E' to pick it up. Press 'Q' once. Press 'Q' multiple times.

Input Specification: 'W'/'A'/'S'/'D'/'E'/'Q' keyboard presses.

Output Specifications: Once the two items have been picked up by the player and the player presses 'Q' once, the player sprite should swap to the one that shows the player holding the first item they picked up. When the player presses 'Q' multiple times, the player sprite should keep swapping between the one that holds the first item and the one that holds the second item, and this swapping should continue until the user stops repeatedly pressing 'Q'.

Pass/Fail Criteria: If the expected behavior laid out in "Output Specifications" is observed, the test passes. Otherwise, the test fails.

#12 - Item-Obstacle Interactions

Description: Items that the player picks up are associated with an obstacle that it interacts with when the player stands next to the obstacle and presses the 'E' key with the item equipped. The obstacle then either disappears, spawns a new item/decoration, or plays an animation.

Items covered by this test: Entity Spawning

Requirements addressed by this test: Requirement 02. When an item is used on its associated obstacle, the feedback (obstacle disappearing/spawning new entities/playing an animation) is roughly instantaneous, preventing player confusion as to whether or not the item is meant for that obstacle.

Environmental needs: N/A

Intercase Dependencies: Test 08, 10, 11. To use an item on its associated

obstacle, the player must first be able to pick up that item wherever it is located and switch to it if need be. The obstacle it is associated with must be able to be spawned into the map.

Test Procedures: Create an item entity. Create an Obstacle entity. In their respective components, pass in the same string for the "tag" parameter (this is how items and obstacles are associated). Place this item and obstacle in the same room. Set the starting room to this room. Start the game. Pick up the item. Navigate to the obstacle it was associated with. When touching the obstacle, press 'E'.

Input Specification: 'W'/'A'/'S'/'D'/'E'/'Q' keyboard presses, and a string "tag" parameter passed into ItemComponent and ObstacleComponent.

Output Specifications: The player sprite should briefly display its "use" animation. The obstacle, depending on what it is, should either disappear, spawn in a new item/decoration, or play an animation.

Pass/Fail Criteria: If the correct feedback/response for the obstacle is observed upon using the item on it, the test passes. Otherwise, if the obstacle does not react correctly or simply remains on screen with no noticeable change, the test fails.

#13 - Escape

Description: When the player leaves the prison, they win.

Items covered by this test: Win Game sequence.

Requirements addressed by this test: Requirement F3.

Environmental needs: N/A

Intercase Dependencies: Requires a successful Tests 1-?? To ensure the player can navigate out of the prison.

Test Procedures: Play the game, escape the prison.

Input Specification: Player movement.

Output Specifications: The "You win" icon.

Pass/Fail Criteria: If the player leaves the prison, and "You win" pops up, pausing the game, test success. Otherwise, the test fails. If there is no way to get to the winning state, the test fails.

#14 - Replayability

Description: Upon winning or losing the game can be restarted, seamlessly.

Items covered by this test: Lose game sequence, win game sequence

Requirements addressed by this test: Test 02, 05. Replay sequence is fast, and should not differ in performance from the first playthrough.

Environmental needs: N/A.

Intercase Dependencies: Test 11 must be successful in order to check the lose game sequence. Test ?? must succeed to check the win game sequence.

Test Procedures: Either get killed by a guard or escape the prison to win/lose the game. Press escape to return to the main menu. Restart the game.

Input Specification: Game is won or lost.

Output Specifications: Game state after pressing play for the second time.

Pass/Fail Criteria: If the game enters an identical start state to the first playthrough, the test succeeds. Any changes are a result of poor memory deallocation and the test fails.

11 Test Results

#00 - Start Screen Navigation

Date(s) of Execution: October 13, 2023

Staff conducting tests: Mitchell, Dylan

Expected Results: Clicking the start game button starts the main game. The how-to-play button displays a list of instructions and controls. The quit button closes the game window.

Actual Results: Clicking the start game button starts the main game. The how-to-play button displays a list of instructions and controls. The quit button closes the game window.

Test Status: Pass

#01 - Entities and Components

Date(s) of Execution: September 15, 2023

Staff conducting tests: Dylan, Daniel

Expected Results: The console displays the old and new positions of the entity.

Actual Results: The console displays the old and new positions of the entity.

Test Status: Pass

#02 - Game sprites

Date(s) of Execution: October 11, 2023

Staff conducting tests: Dua'a, Daniel

Expected Results: The correct sprites are rendered to their correct positions with correct animations if enabled.

Actual Results: The correct sprites are rendered to their correct positions with correct animations if enabled.

Test Status: Pass

#03 - Room-Generation

Date(s) of Execution: October 11, 2023

Staff conducting tests: Dylan, Mitchell

Expected Results: Room is correctly rendered, with the player only colliding with desired walls.

Actual Results: Given a new room with the correct dimensions and tiles according to its text file, the room renders correctly. Collision also works as expected.

Test Status: Pass

#04 - Room-Map Management

Date(s) of Execution: October 23, 2023 [Pass], November 3, 2023 [Fail], November 10, 2023 [Pass]

Staff conducting tests: Dylan, Mitchell

Expected Results: The player leaves the current room and ends up in the correct room and location, and this works for all cardinal directions.

Actual Results: [10/23/2023] Initial room switching play test successful between two rooms. [11/3/2023] After all rooms had been completed and inserted into the map, an issue was discovered where the room transition would occur twice in a row upon entering or leaving certain rooms. [11/10/2023] After a potential patch had been applied to the code, a second play test was run and room switching worked as intended for all rooms and all directions.

Test Status: [10/23/2023] Pass, [11/3/2023] Fail, [11/10/2023] Pass

#05 - Colliders

Date(s) of Execution: October 2, 2023 [Fail], October 6, 2023 [Fail], October 9, 2023 [Pass]

Staff conducting tests: Dylan, Mitchell, Daniel

Expected Results: The player stops moving when reaching the wall, and if inputting directions while in contact with the wall, glides along it without clipping through it.

Actual Results: [10/2/2023] The player stops moving when reaching the wall, but the collision is not smooth and cannot glide along the walls. [10/6/2023] The player stops moving when reaching the wall and is able to glide along it only in certain directions, clipping through it in other directions. [10/9/23] The player stops moving when reaching the wall, and if inputting directions while in contact with the wall, glides along it without clipping through it.

Test Status: [10/2/2023] Fail, [10/6/2023] Fail, [10/9/23] Pass

#06 - Guard Pathfinding

Date(s) of Execution: November 9, 2023

Staff conducting tests: Dylan

Expected Results: The guard follows the player.

Actual Results: The guard follows the player.

Test Status: Pass

#07 - Guards Kill Player

Date(s) of Execution: November 9, 2023 [Fail], November 16, 2023 [Pass], November 17, 2023 [Pass]

Staff conducting tests: Dylan, Mitchell

Expected Results: The guard kills the player when they are right in front of them.

Actual Results: [11/9/2023] While the guard has the ability to kill the player, he did so either not facing the player or being within line of sight but too far away from the player. [11/16/2023] Upon starting a brand new play test, the guard eventually spots the player and kills them if the player remains still. [11/17/2023] Subsequent tests involving different idle player positions yielded the same results. If walking up to a guard, the guard will also kill the player and end the game.

Test Status: [11/9/2023] Fail, [11/16/2023] Pass, [11/17/2023] Pass

#08 - Entity Spawning

Date(s) of Execution: October 13, 2023

Staff conducting tests: Dylan, Mitchell, Daniel, Dua'a

Expected Results: The entity appears in the correct room in the correct location, with the correct properties given to it through ItemComponent, ObstacleComponent, or ObjectComponent.

Actual Results: When adding an item, object or obstacle to the game, it appears in the correct location. This was proved by calculating its position to be placed beforehand based on the map size and tile size. Based on this estimation, the items all appeared where expected.

Test Status: Pass

#09 - Player Movement

Date(s) of Execution: September 18, 2023

Staff conducting tests: Daniel, Dylan

Expected Results: When 'W' is pressed but not held down, the player moves up (north) only slightly and then stops. When 'A' is pressed but not held down, the player moves left (west) only slightly and then stops. When 'S' is pressed but not held down, the player moves down (south) only slightly and then stops. When 'D' is pressed but not held down, the player moves right (east) only slightly and then stops. When 'W' is pressed and held down for a few seconds, the player moves up (north) for the duration of the keypress and then stops. When 'A' is pressed and held down for a few seconds, the player moves left (west) for the duration of the keypress and then stops. When 'S' is pressed and held down for a few seconds, the player moves down (south) for the duration of the keypress and then stops. When 'D' is pressed and held down for a few seconds, the player moves left (east) for the duration of the keypress and then stops. When 'W' and 'A' are pressed simultaneously, the player moves northwest. When 'W' and 'D' are pressed simultaneously, the player moves northeast. When 'S' and 'A' are pressed simultaneously, the player moves southwest. When 'S' and 'D' are pressed simultaneously, the player moves southeast.

Actual Results: When 'W' is pressed but not held down, the player moves up (north) only slightly and then stops. When 'A' is pressed but not held down, the player moves left (west) only slightly and then stops. When 'S' is pressed but not held down, the player moves down (south) only slightly and then stops. When 'D' is pressed but not held down, the player moves right (east) only slightly and then stops. When 'W' is pressed and held down for a few seconds, the player moves up (north) for the duration of the keypress and then stops. When 'A' is pressed and

held down for a few seconds, the player moves left (west) for the duration of the keypress and then stops. When ‘S’ is pressed and held down for a few seconds, the player moves down (south) for the duration of the keypress and then stops. When ‘D’ is pressed and held down for a few seconds, the player moves left (east) for the duration of the keypress and then stops. When ‘W’ and ‘A’ are pressed simultaneously, the player moves northwest. When ‘W’ and ‘D’ are pressed simultaneously, the player moves northeast. When ‘S’ and ‘A’ are pressed simultaneously, the player moves southwest. When ‘S’ and ‘D’ are pressed simultaneously, the player moves southeast.

Test Status: Pass

#10 - Item Pickup

Date(s) of Execution: November 2, 2023

Staff conducting tests: Dylan, Daniel, Dua’a

Expected Results: When ‘E’ is pressed when the player is on top of the item, the item disappears and the player sprite changes to one where the player is holding the item that was just picked up.

Actual Results: When ‘E’ is pressed when the player is on top of the item, the item disappears and the player sprite changes to one where the player is holding the item that was just picked up.

Test Status: Pass

#11 - Item Switching

Date(s) of Execution: November 4, 2023

Staff conducting tests: Dylan, Mitchell, Dua’a

Expected Results: Once the two items have been picked up by the player and the player presses ‘Q’ once, the player sprite swaps to the one that shows the player holding the first item they picked up. When the player presses ‘Q’ multiple times, the player sprite keeps swapping between the one that holds the first item and the one that holds the second item, and this swapping continues until the user stops repeatedly pressing ‘Q’.

Actual Results: Player item swapping works as expected following several in depth play tests of a whole run through the game. While holding an item and picking up another one, pressing ‘Q’ switches back to the old item. Pressing ‘Q’ successfully swaps between items, without fail or odd visual glitches.

Test Status: Pass

#12 - Item-Obstacle Interactions

Date(s) of Execution: November 2, 2023

Staff conducting tests: Dylan, Daniel, Mitchell

Expected Results: The player sprite briefly displays its "use" animation. The obstacle, depending on what it is, either disappears, spawns in a new item/decoration, or plays an animation.

Actual Results: When using an item, the associated animation plays. This same result was found throughout extensive play testing of all items. When using the item on an obstacle, the obstacle was either successfully destroyed (no longer visible) or visible (in an altered state) after interacting with it.

Test Status: Pass

#13 - Escape

Date(s) of Execution: November 16, 2023 [Pass], November 17, 2023 [Pass]

Staff conducting tests: Dylan, Mitchell

Expected Results: The player leaves the prison, and "You win" pops up, pausing the game.

Actual Results: [11/16/2023] Upon successfully blowing up the escape room wall and walking outside the prison, the "You Won" asset successfully displays as an overlay to the outside prison background. The game loop is also paused until the player closes the window or goes back to the start screen. [11/17/2023] A second test was performed by another group member, yielding the same results as the previous test.

Test Status: [11/16/2023] Pass, [11/17/2023] Pass

#14 - Replayability

Date(s) of Execution: November 17, 2023 [Fail], November 21, 2023 [Pass]

Staff conducting tests: Dylan, Mitchell

Expected Results: The game enters an identical start state to the first playthrough.

Actual Results: [11/17/2023] Initial play tests for replayability were conducted and while the player can successfully complete the game or die to a guard, the game would enter a severely buggy state when choosing another play through. For example, visual glitches occurred with the renderer on Mac. For both Mac and PC, the game would crash when entering another room. After extensive code inspection

and debugging, the issues were identified and fixed. Following the fixes, another play test was performed. [11/21/2023] The game successfully entered a fresh state upon starting another play through of the game. During testing, the game was able to be played through in its entirety multiple times, with both winning and dying to the guards.

Test Status: [11/17/2023] Fail, [11/21/2023] Pass

12 Regression Testing

All tests 00-14 were functionality tests that were repeated as new features were implemented into the game. As new features were added, playtesting was done to test that new functionality, and in turn, by playing the game, all tests of previously-implemented functionalities were "repeated" as well. That is, by playtesting the game to verify a new functionality, we also were able to observe whether previously-implemented features broke as a result of the change.

IV Inspection

13 Items to be Inspected

Classes

Inspect constructors and destructors of the various classes used so that variables and data structures are properly initialized and destroyed throughout the run of the game instance.

General Codebase

Look throughout the various code related game files and make sure that good coding practices are followed. Check that variable names are short, descriptive and consistent across all .cpp and .h files. Refactor the code to remove any global variables that may be problematic in the future. Any pointer variables are initialized to null.

Dynamic Memory Allocation

Identify areas in the code base that perform dynamic memory allocation. Investigate whether it is truly necessary. If so, use good practices to avoid memory leaks such as making sure newly allocated memory is deleted somewhere in the code. Assign pointers referring to deleted memory back to null value.

14 Inspection Procedures

Our investigation procedure involved looking over the items listed above during and outside of meetings. The team normally met on Monday, Wednesday and Friday for at least an hour. During these in person meetings, inspections were performed when needed, such as a group member proposing a new addition into the master branch, hosted by GitHub.

Occasionally, group members would also communicate electronically using Discord to solicit and give feedback regarding potential code or asset additions to the game. However, the group as a whole did their main communication during the frequent in person meetings.

Once a significant addition to the game was created, such as a new class, it was reviewed by other members of the group. Key things looked over were functionality and naming conventions used.

15 Inspection Results

Main

With the creation of the game start screen, changes to main.cpp were made by Mitchell in order for the game to support the display of the start screen. These changes were reviewed by Dylan on September 15, 2023. No flaws were identified.

Collision

On October 5, 2023, Mitchell submitted his code to handle collision between walls within the game. The following day, it was inspected by Dylan. While the collision did work for walls in all directions, Dylan noted the complexity and brute force methods used in this approach. He also noted how it would make building maps more difficult in the future.

On October 9, 2023, Dylan proposed his own solution to the game collision that used a more mathematical approach. The code was reviewed by Mitchell, who did not identify any obvious issues with the code or methods used.

Pause functionality

Mitchell was tasked with adding pause functionality to the game, particularly in the update function within Game.cpp. These changes were reviewed by Dylan during the in person meeting on October 11, 2023. No flaws were identified in the approach used.

MapManager

Initial inspection occurred when the class had been written by Mitchell during the meeting on October 13, 2023. Dylan inspected it, particularly what data structure was going to be used to store the map rooms and what parameters were needed for the room change function. No immediate flaws were identified during this inspection.

A second inspection was performed by Dylan on November 3, 2023. This inspection was preceded by a playtest of the game, using the MapManager class for room traversal. Dylan identified that the addRoom function was missing a second condition check in order to break out of a loop while adding a room into an array. He then proposed a fix to this function.

Shortly after, a follow up inspection was performed on the same day by Mitchell regarding the potential fix. It was deemed fit for use and merged into the master branch.

Game over and game win

On November 15, 2023, Dylan had added additions to Game.cpp to support the game over and game win states. Since Mitchell was tasked with creating the image asset pop ups for these stages, he also performed the code inspection. No issues were identified with the addition of these changes into the game loop update function.

Final code inspection

On November 21, 2023, a final review of the code base was performed by Mitchell in preparation for the final game demonstration. This was in response to issues related to crashes and memory management being identified during play testing, particularly during consecutive playthroughs of the game.

Mitchell identified the issue as being related to the large number of global variables present in the Game.cpp file. These variables always remained in scope, resulting in destructors not being called and various glitches during a second playthrough regarding game rendering and room switching.

Another issue identified in this inspection was improper usage of destructors, particularly in the MapManager class and Room class. Not all dynamically allocated variables were properly deleted and pointers set to null.

The following morning, on November 22, the group met and Dylan performed a review of the optimization related fixes to the overall game. No obvious issues were seen in the revisions, and they were determined to be good for use.

V Recommendations and Conclusions

Items covered:

- Start Screen navigation [testing and inspection process passed]
- Room-generation [testing and inspection process passed]
- Map-generation [testing and inspection process passed]
- Room-switching [testing and inspection process passed]
- Entity components [testing and inspection process passed]
- Collision detection [testing and inspection process passed]
- Displaying sprites [testing and inspection process passed]
- Sprite-switching [testing and inspection process passed]
- Sprite animations [testing and inspection process passed]
- Player/Item/Obstacle/Decoration spawning [testing and inspection process passed]
- Player movement [testing and inspection process passed]

- Adding items/obstacles/decorations to room (required Player/Item/Obstacle/ Decoration spawning) [testing and inspection process passed]
- Picking up items [testing and inspection process passed]
- Item-switching [testing and inspection process passed]
- Interactions between items and obstacles [testing and inspection process passed]
- Guard AI pathfinding [testing and inspection process passed]
- Lose game sequence [testing and inspection process passed]
- Win game sequence [testing and inspection process passed]

Actions to take next:

- Conduct tests on the game using different window sizes as this version only was tested to support a specific window size
- Conduct a test with the game window set to full screen to ensure it displays properly
- Implement a dedicated/more sophisticated “You Win” and “You Lose” screen for when the player escapes the prison or gets shot by the guard, with buttons to go back to the start screen.
- Implement a crate/barrel explosion animation for the ending sequence
- Implement item-use animations longer than one frame
- Implement a graphical inventory menu that can be pulled up, showing all items the player has collected so far, rather than the player having to swap between items in order
- Implement a minimap displaying rooms that the player has visited so far
- Allow the player to equip items when wearing the guard disguise

VI Project Issues

16 Open Issues

Towards the end of the development process, after certain features and changes have been implemented to account for multiple, simultaneous play throughs, an issue was identified related to the performance of the game. Specifically, it was observed by one of our play testers that there was an increase in game stuttering or lag after dying to an NPC or winning the game and starting another run through the game. This issue is mostly relevant to section 12a from the development specification [1].

17 Waiting Room

There are a few features that the team did not get the chance to apply to the current release of the project. The current development team were unable to implement the feature of multiplayer with server and client functionality with the current version, and the ability to track how long it takes for the player to complete their escape, with a ranking system to keep track of the fastest times. However, with the current status of Prison Break, it is relatively easy to accommodate. These features would have allowed for the player to play the game with others that would allow for fun, unique game play

experiences.

As for requirements, the team would have liked to add the random map requirement that was documented in the original design to allow for replayability value [1]. The original team's design was to have Prison Break be a multiplayer ordeal where players would be able to work together in order to escape the prison, but the current version released has only been able to support single player. With the current state of the game, it would be relatively straightforward to add this feature, but due to time constraints, it would have to be in the next release that this feature is added.

Finally, the game is currently lacking a player selection screen with statistics unique to each player, a display of the current character coordinates within the map, mobile device support and the ability to share game saves between mobile and desktop per the design specification [1]. As with the prior pending feature implementations, most of these would be somewhat trivial to implement with more development time. The only features that would take some significant work would be adding support for mobile devices along with the account system to facilitate the transfer of save data between devices.

18 Ideas for Solutions

One of the situations that was presented when it came to developing Prison Break was the libraries that would be used to allow for development to be independent from the devices that it would be created and developed on, in which the solution that the team concluded to use was the use of the SDL Libraries sourced from libsdl.org. Using the SDL libraries allowed the development team to create a common ground to work together due to the differences in devices and allowed the team to work on the game without conflicts. For the current development up until the current version, the solution of using the SDL libraries was a successful solution.

However, when looking to expand the features of the game and scale it up, there are concerns about the feasibility of sticking with SDL as the game library of choice. First off, SDL is moreso a library of core functionalities that prove useful to game creation rather than a genuine game engine. Second, SDL is limited in the functionalities it provides and the functionalities it does provide are fairly basic. Third, for more sophisticated enhancements to the game (like 3D rendering), developing with SDL is far too complicated and a game engine like Unreal Engine would be better suited for the task. With all these considerations in mind, it is in our best interest to migrate development to the Unreal Engine, where the more advanced and sophisticated libraries and tools it provides would make the process of adding significant overhauls to Prison Break much easier.

19 Project Retrospective

For the development of Prison Break, something that proved to be a benefit was the team meeting frequently in order to coordinate between the different aspects of the game that were being developed from all contributors. Any debugging issues or

improvements to sprites and gameplay were able to be discussed face to face to achieve a suitable solution and begin implementation of said solution without as big of a delay, which would not have been possible if the team met only once a week.

Some points in development that proved to be more difficult to deal with or could have been improved to some degree is the consistency of the sprites across the game. In order to ensure the game remained uniform, the sprites that were created such as the items, the player, and the NPCs, were meant to be as consistent as possible. However, due to the requirement of visibility, there was some debate as to how the best visibility could be achieved, and it led to several iterations of sprite edits to see how they would appear on screen during gameplay before the group decided which version was the most consistent and most visible.

Another point in development that could have been improved was having a shared folder of assets such as decorations, items, and obstacles from the beginning, as well as keeping a list of these assets and what each purpose was. The team did not have access to such a list, and it resulted in delays with completing sprites when that time could have been spent more on implementation of other aspects of the game. Having the documentation of what the development team needed prior would have allowed for more time to be accounted for creating rooms, organizing the layout of the map, and implementing more functionality that otherwise did not make it into the current version of the game.

VII Glossary

Class: A set of functions and variables associated with a particular object.

Collision: Detecting when a player interacts with items and walls within the game environment.

Component: Any characteristic that can be attached to an entity by means of a .h file extending the Component class. These would include an entity's transform component (location/size of entity), sprite component (visual representation of the entity), collision component (the ability to collide with other entities who possess this component), etc. Entities can have the same components as other entities but each instance of a component is unique to that particular entity.

Entity: Any physical object in the game that can be placed in a room and possess characteristics, such as the player, items, obstacles, decorative objects, room-tiles, collisions-tiles, and guards.

GitHub: A website that allows software developers to store and manage versions of their code.

Item: A component of an entity that allows it to be picked up by the player as well as possess a unique id

Map: An array of rooms which make up the prison as a whole. The map allows the exiting of one room to return an instance of another, so that the player can make their way throughout the prison.

Navmesh: The “Navigation Mesh” method of pathfinding used to supply the NPCs with paths to move throughout their rooms. Prison Break uses a set of nodes generated from room text files and Dijkstra’s shortest path algorithm in its Navmesh class in order to create smooth AI movement. While not a true “Navmesh” as seen in popular game engines like Unreal or Unity, the navmeshes seen in these engines served as inspiration for AI navigation as seen in Prison Break.

NPC: A component that allows an entity to follow a path of nodes supplied to it. In Prison Break, the main NPCs are prison guards.

Object: A generic characteristic component of an entity having both a transform and sprite component.

Obstacle: A component of an entity that makes the entity restrict the player’s movement and be removed from the room by the player if he/she is holding an item with an id matching the obstacle’s key variable.

Room: An instance of the Room class which possesses a file path to tilemap, containing both the locations of tiles to be rendered to the screen as well as information like the location of wall “colliders” and Navmesh nodes. It also possesses a list of objects to be loaded upon the player entering the room, and two dimensional vectors containing the locations of each entrance/exit of the room, one for each cardinal direction.

SDL: Simple DirectMedia Layer. A cross platform development library that provides low level access to computer hardware.

Sprite: A 2D image used in game development to represent a character or object.

Tile: A square image displayed in game. When combined with other tiles in a grid, it makes up the background of the game.

Tileset: A single image containing all of the tiles to be displayed within the game

VIII References / Bibliography

[1] S. Yadavalli, H. Hussain, T. Castino, J. Peterson, “Prison Break Project Report”, 2019.

[2] Robertson and Robertson, Mastering the Requirements Process.

[3] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, Ninth ed., Wiley, 2013.

[4] J. Bell, "Underwater Archaeological Survey Report Template: A Sample Document for Generating Consistent Professional Reports," Underwater Archaeological Society of Chicago, Chicago, 2012.

[5] M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004.

IX Index

actual 13, 24, 25, 26, 27, 28, 29
break 4, 5, 7, 8, 9, 10, 13, 14, 17, 31, 33, 34, 36
button 14, 15, 24
class 4, 5, 8, 13, 16, 17, 18, 19, 31, 32, 35, 36
code 10, 13, 25, 29, 30, 31, 32, 35
collision 4, 7, 14, 18, 25, 26, 31, 32, 35
component 4, 5, 13, 15, 16, 18, 19, 35, 36
components 4, 8, 13, 14, 15, 16, 19, 20, 23, 24, 32, 35
correct 16, 17, 19, 20, 23, 25, 27
current 4, 13, 14, 16, 17, 18, 25, 33, 34, 35
dependencies 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
description 4, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23
development 4, 33, 34, 35, 36
down 20, 21, 27, 28
entities 4, 13, 14, 15, 16, 17, 18, 19, 22, 24, 35
entity 4, 5, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 27, 32, 35, 36
environmental 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
escape 4, 8, 10, 13, 23, 24, 29, 33, 34
execution 24, 25, 26, 27, 28, 29
expected 20, 21, 22, 24, 25, 26, 27, 28, 29
fail 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30
fails 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
figure 6, 7, 8, 9, 10, 11, 12, 13
functionality 4, 7, 8, 10, 13, 14, 30, 31, 33, 35
game 4, 5, 6, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36
guard 10, 11, 14, 18, 19, 24, 26, 29, 33
guards 4, 5, 10, 18, 26, 30, 35, 36
held 20, 21, 27, 28
input 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
inspection 29, 30, 31, 32, 33
item 5, 8, 14, 21, 22, 23, 27, 28, 29, 32, 33, 35, 36
items 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 27, 28, 29, 30, 32, 33, 35
location 4, 5, 17, 18, 19, 20, 25, 27, 35, 36
lose 10, 14, 19, 23, 24, 33

map 4, 5, 7, 8, 10, 14, 17, 23, 25, 27, 31, 34, 35, 36
move 5, 10, 17, 18, 20, 21, 36
movement 5, 7, 14, 18, 19, 20, 23, 27, 32, 36
moves 27, 28
new 8, 14, 15, 16, 17, 22, 23, 24, 25, 26, 29, 30, 31
obstacle 5, 8, 14, 22, 23, 27, 29, 32, 33, 36
obstacles 4, 6, 8, 10, 13, 14, 19, 23, 33, 35
output 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
pass 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
passed 20, 23, 32, 33
play 14, 23, 24, 25, 26, 28, 29, 30, 32, 33
player 4, 5, 7, 8, 9, 10, 11, 12, 14, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36
pressed 20, 21, 27, 28
prison 4, 5, 7, 8, 9, 10, 12, 13, 14, 17, 23, 24, 29, 33, 34, 36
procedures 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 30
process 20, 32, 33, 34, 36
project 4, 6, 13, 14, 33, 34, 36
release 7, 8, 9, 10, 13, 17, 20, 33, 34
requirement 15, 17, 19, 20, 21, 22, 23, 34, 35
requirements 4, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 34, 36
results 24, 25, 26, 27, 28, 29, 31
room 4, 5, 6, 7, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 29, 31, 32, 33, 35, 36
rooms 4, 5, 8, 10, 14, 17, 19, 21, 22, 25, 31, 33, 35, 36
screen 5, 14, 15, 23, 24, 29, 31, 32, 33, 34, 35, 36
specification 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 33, 34
specifications 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
sprite 4, 7, 14, 16, 19, 20, 21, 22, 23, 28, 29, 32, 35, 36
sprites 7, 9, 14, 16, 19, 25, 32, 35
start 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 29, 31, 32, 33
stops 18, 22, 26, 27, 28
test 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 33
testing 14, 19, 29, 30, 32, 33
tests 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 33
used 4, 5, 6, 7, 10, 22, 30, 31, 34, 36
win 14, 23, 24, 29, 32, 33