CIS*3250: Assignment 1
October 23, 2015

Sanchitt Dhir
Justin Galvez
Filip Hasson
Antonio Jajou
Conner Keating
Daniel Miller
Marc Santos
Jessy Williams-Rancourt
Karol Zdebel

RIOT: A REVERSE TOWER DEFENCE GAME

DESIGN & REQUIREMENTS SPECIFICATION DOCUMENT

TABLE OF CONTENTS

1. PREAMBLE

1.1 GAME OVERVIEW

RIOT is a reverse tower defence game set in a prison theme. It is a turn based strategy game where the player is tasked with breaking out of correctional facilities by recruiting and deploying waves of inmates against statically positioned guards with the intention of generating panic and making their escape. RIOT intends to switch the roles of a traditional tower defence game, placing the user in control of what would conventionally be the enemy horde.

1.2 TARGET AUDIENCE

The target audience for RIOT is gamers aged 14-25 who enjoy strategy and casual games-- largely the same demographic who enjoy traditional tower defence games. RIOT will attract players looking for something new in the genre by introducing a unique setting and will appear to users who enjoy modding by allowing user created scenarios (see 6.2).

2. GAMEPLAY MECHANICS

2.1 PLAYER OBJECTIVE

The user is tasked with sending waves of inmates (see 2.8) through the gauntlet of guards (2.7) and obstacles (3.3) presented in each level in order to build up enough panic (2.3) to make their escape.

2.2 WAVES

RIOT features turn based gameplay. During the player's turn they are able to create a queue of up to 5 inmates (see 2.9). These units make up a wave which is released once the player has confirmed their selection by pressing return. At this point the game switches into a real-time mode which processes the inmates' journey through the level. Once the last unit of the wave has been busted or has escaped, control is returned to the user. These two modes of gameplay will continue until the conclusion of the level.

2.3 PANIC

The player builds up panic in a level by successfully leading an inmate from the start position through to the end position of a level. Each level has a panic threshold presented to the player as a percentage and each inmate type has a set amount of panic that they can generate (see 2.8), which will contribute to this meter as they successfully exit the level. Guard accuracy is inverse to the panic percentage. For instance, if the panic level is 0% the guards have 100% accuracy, or conversely, if the panic level reaches 100%, guard have an accuracy of 0%.

2.4 REP

As the player progresses through the game and successfully lead their escapes, they will earn reputation or 'rep' for short. Rep is a currency in

the game which can be used to recruit inmates (2.8) and a set amount is provided at the beginning of each level.

2.5 TIME (CYCLES)

The base unit of time upon which inmate movement and guard attack cool-downs are based is referred to as a cycle. This measure of time is also the basis for updating the state of the game board and updating the display (see 5.2.3). A cycle is tentatively set to 1/6th of a second, however this will need to be revisited while adjusting gameplay balance during development (7.3).

2.6 PROGRESSION

RIOT will ship will 9 levels (see 3.1), each of which take place in a different detainment facility. Levels will be unique and range from a classroom to a supermax prison. As the player progresses through the game, new obstacles will be introduced (3.3) , along new guards (2.7) and inmates (2.8). Rep (2.4) will increase, however will will be allotted less generously, requiring greater resource management by the player. This combined with an increase in possible strategy permutations will scale level difficulty.

2.7 GUARDS (TOWERS)

Guards represent the towers of a traditional tower defence game. They are stationary within each level and cannot be injured or removed during gameplay. Guards are represented by uppercase characters on the game board in order to distinguish them from inmate units (2.8), which use lowercase characters. Guards have 3 main stats: damage, range, and cool-down. Damage refers to how much damage they will inflict to a target per cycle. Range refers to the maximum distance that they can target an inmate from. Cool-down refers to how many cycles a unit will wait before attacking again.

Name	Damage	Range	Cool-down	Targetting	Ability/ Comment
[**G**]uard	5	2	4	Proximity	
[**D**]ogs	4	4	6	Area of effect	Doubles further damage; cannot be bribed
[**L**]unchlady	0	6	12	Area of effect	Inmate speed halved for 12 cycles
[**P**]sychiatrist	0	6	12	Proximity	Puts an inmate to sleep for 6 cycles
[**S**]harpshooter	6	10	8	Closest to exit	Can reach long distances; cannot be bribed
[**W**]arden	100	2	12	Proximity	Busts any adjacent unit; cannot be bribed
[**C**]yborg	12	8	2	Proximity	Available on LunarMax; cannot be bribed

Fig A: Guard Statistics and Traits

2.7.1 ATTACKING

Guards also have a particular behaviour profile which determined how they determine their targets, and some have a special ability unique to the unit. Some units can be bribed by particular inmates (Fig.B), which prevents them from attacking the next inmate to pass. When a guard attacks, this is indicated by inverting the guard character’s background and foreground colours. Projectiles are not visualized on screen in order to avoid congesting the limited resolution (5.1).

2.8 INMATES (ENEMY HORDE)

Inmates are deployed by the player in waves and represent the enemy horde of a traditional tower defence game. Inmate sympathizers such as the attorney and doctor will also be referred to henceforth as inmates. Each inmate unit has 4 main stats: health, speed, rep, and panic. Health points determine how much damage an inmate can take before getting busted. Once busted, a unit will be removed from the game. Player health is conveyed to the player using colour. Units with 100-75% of their initial health are green, with 74-50% are yellow, with 49-25% are orange, and lastly with 24-1% are red. Movement speed refers to how many spaces an inmate moves per cycle (see 2.5). Panic

refers to the amount of panic an inmate contributes once it has successfully made it through a level (2.3).

Name	Health	Speed	Rep	Panic	Ability/ Comment
[**r**]un (protagonist)	5	2	0	0	The game's unnamed protagonist; must escape to complete the level
[**h**]omeboy	10	4	5	2	
[**b**]ruiser	16	4	15	6	
[**l**]unatic	16	6	10	8	Attacks adjacent inmates
[**f**]atty	40	2	10	4	'Tank' unit
[**s**]peedy	10	8	20	2	Increases adjacent inmate speed to by 2, up to a maximum of 6
[**c**]utie	20	4	20	1	Disables guard attacks within 6 tiles
[**a**]ttorney	30	4	30	2	Bribes guards within 6 tiles
[**d**]octor	10	5	40	2	Heals adjacent inmates 1 hp/ cycle

Fig B: Inmate Statistics and Traits

2.8.1 INMATE MOVEMENT

When a wave of inmates is released, units will begin to traverse the map by following the game path (see 3.2). Units cannot pass one another and will be bottlenecked by any other further units blocking the path. Unit speed is relative to a cycle (2.5).

2.8.2 PROTAGONIST UNIT

At any point the user can send the unnamed protagonist inmate by pressing the [r] hotkey. This unit has minimal health and can be busted by most guards, emphasizing the importance of strategically deploying supporting inmates.

2.9 QUEUE

Players can send up to 5 inmates through the level at a time. During inmate selection the game is in a paused state. The queue is illustrated to the

right of the map and displays the current selected units by displaying their associated hotkey (see Fig B).

2.10 LEVEL COMPLETION

Should the user be able to reach the exit with the protagonist inmate, they will have completed the level, be presented with a story message (see 6.2), be brought to the continue menu and unlock the following level (4.3). If the protagonist user is busted however, the user will be notified, also returned to the continue menu, however will not unlock the following level. Once the user has completed the final available, they will have beaten the game and presented with an endgame message.

3. ENVIRONMENTS

3.1 MAPS

In the initial release of the game there are 9 levels. Level layouts are stored in external text files (see 6.2). These files can be added to or extended by either the user or development team at a later date however.

3.1.1 THE CLASSROOM (TUTORIAL)

The first level takes place in a classroom. Fitting with the classroom theme, this level will serve as a tutorial for the player. There are only two possible outcomes-- either the user sends a homeboy or the protagonist. If the user sends the protagonist first they will get busted. If the inmates sends the homeboy first, which has an extra health point,, this will allow the protagonist to get through after. This implicitly teaches the user the core mechanics of the game. If the user fails they will be informed explicitly about their mistake.

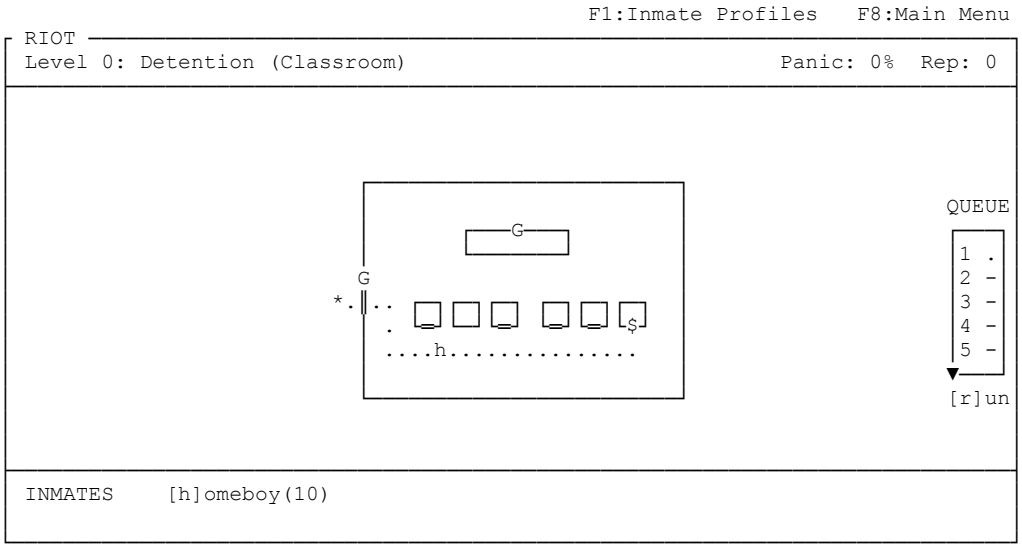


Fig C: A mockup of the tutorial level

3.1.2 PRISONS

The core game will include 7 prison levels including the following:

- The Drunk Tank
- Minimum Security Facility
- Tijuana Lockup
- Medium Security Facility
- Supermax
- Guantanamo

Each level will introduce a new unit (see 2.8), along with new guard units (2.7) and obstacles (3.3). Levels are all visible from the continue screen, however remain locked until the previous level has been completed.

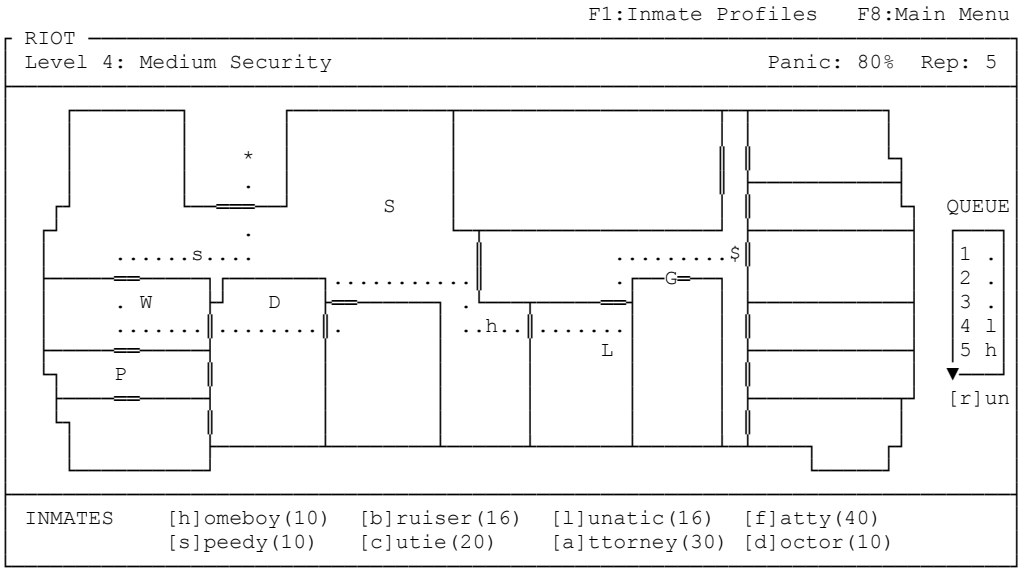


Fig D: A mockup of a prison level

3.1.9 LUNARMAX (BONUS LEVEL)

As a reward for the user having beaten the main game, they will unlock Lunarmax, a space prison level. This level will be hidden from the continue menu until they have successfully completed Guantanamo. The level has zero

gravity, where inmates move at half speed, and unlocks the Cyborg guard, a powerful endgame unit.

3.2 THE PATH

Each level will contain a single path to be traversed by player units. The starting point of the path is indicated by the dollar sign character ($) and terminated with the ampersand character (&). The path itself is drawn using the period character (.). Inmates travel down the path at a rate of one character per cycle (see 2.5).

3.3 OBSTACLES

As the game progresses, different sections of the path may have different traits which provide an additional challenge. Doors are illustrated using hashtag the hashtag character (#) and require 5 cycles to unlock. Doors will remain unlocked for subsequent waves. There are also parts of the path in which player movement is reduced by half-- for instance over mud outside or in zero gravity, which is indicated by the percentage sign character (%).

4. USER INTERFACE

4.1 OVERVIEW

The program can be executed and ran within a terminal window with a size of 80x24 characters. The game runs in curses mode and makes use of UTF-8 characters which which will be required to be supported for the optimum user experience. Terminals which do not support UTF-8 characters may default to backup ASCII character sets, however this is platform dependant.

4.2 DESIGN RULES

The two main priorities in the game’s design are consistent presentation and intuitive controls.

Both the main menu and the game screen are required to fit inside the 80x24 window and are to be outlined with a border. Every other screen is required to prominently display the game’s title in order to enforce the game’s brand. In every context and menu, the game is controlled and navigated by pressing the lowercase alpha keys as indicated by square brackets around hotkey characters.

4.3 FRONT END

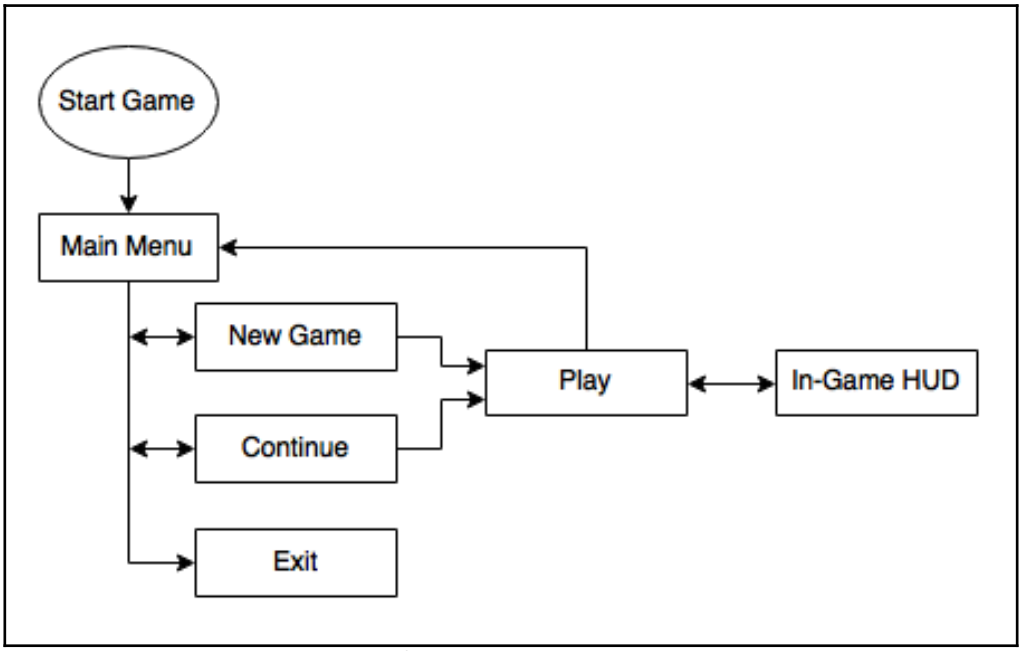


Fig E: Menu Flow

4.3.1 MAIN MENU

```
8888888b. 8888888 .d88888b.88888888888
888   Y88b  888  d88P" "Y88b   888
888    888  888  888     888   888
888   d88P  888  888     888   888
8888888P"   888  888     888   888
888 T88b    888  888     888   888
888  T88b   888  Y88b. .d88P   888
888   T88b8888888 "Y88888P"    888

_____________________________________


GAME MENU

[n]ew game
[c]ontinue
```

Fig F: Mockup of the main menu

4.3.2 NEW GAME

New game will set the lock flag in all loaded map files (see 6.2) and automatically execute the first first map.

4.3.3 CONTINUE

Continue will read the lock status of all maps and present them to the user in an enumerated ordered list. Level numbers will be hotkeyed to the level's corresponding number.

4.3.4 IN-GAME HUD

In addition to the inmate hotkeys (see Fig.B), the user can press F1 to open inmate profiles, or F8 to abort the current game and return to the main menu. These keys are spaced far apart to prevent accidental or unintentional keypresses. The inmate profile will provide an additional description and stats (2.8) about inmate units which extend beyond what is presented during gameplay.

4.4 USER INPUT

All user input is given through the keyboard. Whether it be on the main menu or the game screen, all the possible commands are paired with hotkeys which are indicated using square brackets.

5. SOFTWARE DESIGN

5.1 CONSTRAINTS AND RESTRICTIONS

5.1.1 EXECUTING CONCURRENT TASKS IN C

C does not lend itself well to running concurrent tasks. Ncurses cannot wait for user input while processing background data without employing multithreading or providing the illusion of multithreading by alternating between the two tasks continuously. Both of these would have a considerable impact on the complexity of development and testing of the program. With this technical limitation in mind, RIOT will focus on gameplay.

5.1.2 VISUALIZATION

Many of the constraints, limitations, and restrictions in designing and developing this software are tied to nCurses and the way in which information is visually conveyed to the user. Even with a large terminal window, the resolution of the text-mode grid and maximum number of displayable characters pale compared to being able to draw at the pixel level. Furthermore, being limited to printing ASCII and UTF-8 characters itself limits artistic freedom. These factors remove any burden of developing detailed art assets.

While using a model view control model would allow for the game landscape to be scaled and formatted separate from the display, this often cannot be rendered correctly in nCurses due to its limited resolution. Angles lines can become jagged if they aren’t right-angled and contents of cells cannot be properly scaled gracefully. This can be confusing and unappealing as character placement becomes inconsistent or if items overlap. In order to address this issue, the game board will be a static 24x80 size. This size was chosen as it is conventionally the default terminal window size, implying a high level of support, and since it imposes a helpful constraint which will guide the design of maps and placement of UI elements.

The interface of RIOT will strategically use screen real-estate as to only directly display information relevant to the the task at hand, while providing the user the option to access any additional information through an alternative method. For instance, during gameplay, only the player units' hotkeys, along with their relevant rep cost are displayed-- the pieces of information critical to playing the game. However, should the user want to get more information in regards to a particular unit, they can open the unit display sub-screen in order to get access to that information. This preserves screen real estate while still providing access to the optional information which could have been displayed otherwise.

5.2 ARCHITECTURAL DESIGN

5.2.1 DESIGN OVERVIEW

The game will be implemented based around a Model-View Controller (MVC) pattern in order to separate game elements' internal representations from the subsystems which will be used to render them. This decision has been made on the basis of game performance and ease of development. Thinking about the game board in layers, only unit placements need to be processed and drawn each cycle-- the game map only needs to be loaded once and remains static throughout gameplay.

Game logic is divided into four modules: input/ output, unit processing, map processing, and visualization. Each of these modules will be referenced as riotIO, riotUnits, riotMap, and riotUI, respectively.

5.2.2 RIOTIO

riotIO will provide an interface to fileIO which will be used for writing and reading .riot files from disk. These files will include the map layout and three text blocks. These will serve as messages which will be displayed on screen before a level to offer context, when the user completes a level to provided congratulations, or when a user fails a level to provide a hint for the level. Lastly, riotIO will process user input from the keyboard, interpret it, and relay it to other program subroutines in a meaningful way.

5.2.3 RIOTUI

riotUI will provide an interface to draw the to the 4 nCurses windows used in the game, namely ‘menu’, ‘main’, ’header’, and ‘footer’. Menu will display the New Game and Continue Menus, main will display the map area, header will display the level title and F# hotkeys, and footer will display the unit hotkeys (see fig 2.8).

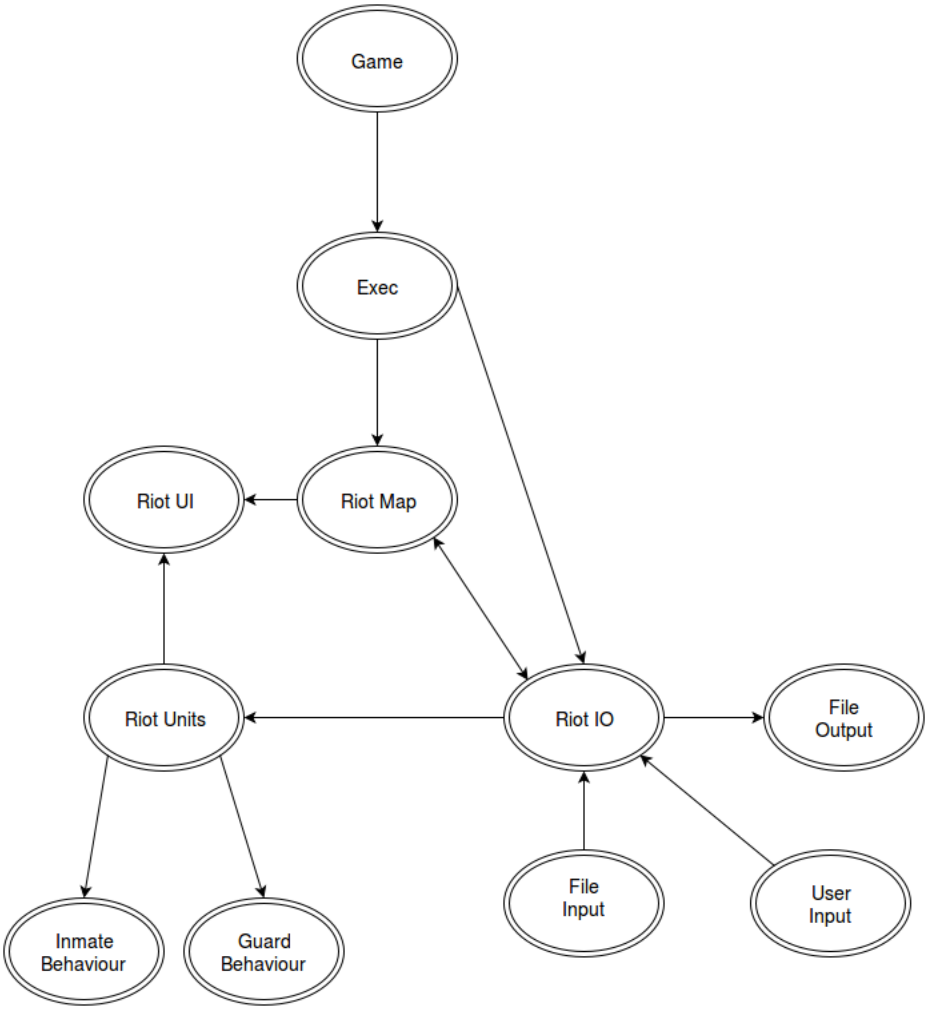
5.2.4 RIOTUNITS

riotUnits will define the characteristics of both inmate and guard units. It will process these unit’s actions and manage the interactions between them. This includes attacking and moving. After each realtime cycle (see 2.5) riotUI will relay changes to be displayed to riotUI.

5.2.5 RIOTMAP

riotMap will interpret map data provided by riotUI and place game elements into appropriate structures (see 6.1).

5.3 ARCHITECTURE DIAGRAM



All game information stems off from the file input where all the level, interface and essential game information is stored. Game Data updates the units(guards and prisoners) all throughout the gameplay. It also receives information from the use(User Input) which contributes to these changes. Game data also communicates what is to be shown to the player(The link from game data to game visuals), and what the player sees is primarily the interface and the units(link from game visuals to interface and units). Finally game data also communicates to the output file which pretty much acts as a save file to be used when the game is resumed.

6. DATA DESIGN

6.1 INTERNAL DATA STRUCTURE

6.1.1 MAP REPRESENTATIONS

The game map will be stored as a 72x16 character array (width x height). this will be parsed from the map files by riotIO and stored within a struct initiated within riotMap.

6.1.2 UNIT REPRESENTATIONS

The game's units will be stored into two types of structs: 'Inmates' and 'Guards'. These will contain the various traits and stats (see 2.8 and 2.9), along with memory reserved for a pointer of the respective type. These will serve as nodes within a pair of linked list data stores.

6.2 STORAGE STRUCTURE

Map files will be parsed and loaded from the disk and then processed into the appropriate internal data structures (see 6.1.1). While this is less efficient than having had hard-coded them into the software, this allows for levels to be added, patched, or expanded by either the developed or users without recompilation of the game binary.

Map files will contain a plain-text map file encoded with 1184 UTF-8 characters, contained in 16 72-character lines (plus a return character). After character 1184, three strings contained within quotations must also be included. These serve as the level dialogue. They may be empty strings, however.

Alternate map source directories will be able to be used by passing directory names as a command line parameter to the RIOT executable. Map files must have a '.riot' extension to be recognized.

7. TESTING

7.1 CLASSES OF TESTS

The type of classes of testing used in this program will be white box and black box testing. The white and black box testing will include: How the unit moves, how the towers will interact with the units(attacking, which to target, etc), how the units will die, each specific units attributes, also loading up each map, how the store will load up, and how the user will be able to purchase from that store.

Unit testing for how the units move will be done by setting a static pathway and seeing if the units will be able to follow it. For the movement part of the testing, the speed of the units has match the flow of the game. The reason for this are that some units should have a different set speed while moving through the levels. This will be tested by timing how long it will take each unit to run through the path.

The next set of testing will be on how the tower will interact with the units. The test that will be conducted are: 1) what are the area of each of the NPC before attacking the units, 2) the attack speed at which the NPC fires at, also the amount of damage they output on the units, and finally 3) their special attributes. The radius will be tested by seeing which targets the guard will hit while there is 1 unit within range at first, then seeing what happens when there are 2 or more unit within range of it. Also, check to see if the tower will only attack units that are within range and are the closest to the guard. There will be unit tests to see if there is a specific firing range each different tower ,should be firing at while interacting with the units and the amount of damage they output.

Loading up each map will also be tested by seeing what happens when the user get a specific amount of units to the end of the path, will it go to the next static map or will it render a completely new one. Numerous tests will be done for the functionality of the store and purchasing units. First, check if the right calculations were implemented when purchasing units from

the store, also making sure they cannot buy units if they do not have sufficient funds for that specific unit, which will also concatenate the values between the store and game.

Unit testing will also need to be conducted on what happens when a unit die. This test is conducted to also see if the game will end properly if there are no more available units to summon. This is important as a game's core objective always has to have a possible bad ending.

Besides unit testing for each individual inmate and guard, unit testing will also be used in the aforementioned features by using incremental integration tests which will start bottom-up i.e. as new features are introduced. This is important because the framework riotIO and then riotMap are needed before anything else can be implemented.

7.2 CRITICAL COMPONENTS

Because the system is a game, almost every part of the system is critical in its use. This being said, the most critical components are not the units or the towers themselves, but would be the source files for RIOT. riotIO is critical in the receiving of the map and game files necessary to display and calculate the algorithms used in the game. We must first ensure the system can take in the file before anything can be showed to the user graphically, and can take inputs to queue units to be sent out.

7.3 GAMEPLAY BALANCING

To test for game balance, many factors need to be looked at. The factors that affect game balance are: unit hp, unit speed, the unit's special characteristics, unit price, tower damage, tower fire rate, tower placement, map design, and map objective. Each unit and tower will have different attributes and stats, so game balance will be difficult as there is a vast amount of combinations when a tower attacks a unit.

At the same time, because of the n component, the difficulty must rise as the player proceeds through the levels, but at an exponential rate rather than a flat one. The player's skill must also be considered when trying to balance the game. The difficulty must be easy enough for first time players to be able to complete at least the tutorial level and a few levels after that, but not easy enough that it is too easy for skilled players. A good way to test for this is to let both developers and non-developers playtest the game. This compliments whitebox and blackbox testing as whitebox testing will be used to do a difficulty check for experienced players, while black box testing will help test the difficulty and see if it is too hard for newer players.

8. APPENDIX

8.1 CONTRIBUTIONS

Group Member	**Contributions**
Sanchitt Dhir	Attended group meetings
Justin Galvez	Target user, testing, game balancing
Filip Hasson	Level descriptions
Antonio Jajou	Testing
Conner Keating	Attended group meetings
Daniel Miller	Unit descriptions
Marc Santos	Unit descriptions
Jessy Williams-Rancourt	Preamble, game overview, target audience, gameplay mechanics, player objective, waves, panic, rep, time, progression, guards, attacking, inmates, level completion, environments, maps, the path, obstacles, user interface overview, design rules, front end, user input, software design, constraints,, architectural design, data design, internal data structure, unit representations, storage structure
Karol Zdebel	Architecture diagram, data design

8.2 CHANGE LOG

Since the original draft document, efforts have been made to improve term consistency, and to fix formatting errors. Passages written in the active voice have been replaced with the passive voice. Duplicate content has been culled. Relevant sections of the document are now referenced by using placing section numbers within parentheses. Formatting grammatical errors have also been addressed. The game’s vision has remained largely similar since the draft, however many mechanics have been elaborated upon and gameplay has incorporated a more strategy oriented style in order to further distance itself from other popular nCurses games which will not be named.