

Group 1

BT

(with major
concerns this
looks like
rogue)

CIS 3250: Assignment 1
October 12, 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

(DRAFT COPY)

- is a post "rogue re-implemented"
- Confused this
 - Format is generally good
 - Some organization needed to better reflect related information in one place
 - lot of detail specifics are missing - length of document is b3pm from strictly necessary due to restriction vague things repeatedly
 - lacks detail for sub systems to permit implementation. The high level stuff presented isn't too bad though - good start

TABLE OF CONTENTS

1. PREAMBLE

- 1.1 Game Overview
- 1.2 Vision Statement
- 1.3 Major Constraints, Limitations and Restrictions

2. GAMEPLAY

- 2.1 Player Objective
- 2.2 Controls
- 2.3 Progression
- 2.4 Guards (Towers)
 - 2.4.1 Properties
 - 2.4.2 Attacking
- 2.5 Inmates (Player Units)
 - 2.5.1 Properties
 - 2.5.2 Movement
- 2.6 Environments
 - 2.6.1 Levels
 - 2.6.2 Path,
 - 2.6.3 Obstacles

3. USER INTERFACE

- 3.1 Design Rules
- 3.2 Front End
 - 3.2.1 Main Menu
 - 3.2.2 New Game
 - 3.2.3 Continue

3.3 Main Game

4. SOFTWARE DESIGN

4.1 Architectural Design

4.1.1 Program Structure

4.1.2 Architecture Diagram

4.2 Data Design

4.2.1 Internal Data Structure

4.2.2 Temporary Data Structure

4.2.3 Storage Structure

4.2.3.1 Map Files

4.2.3.2 Save Files

5. TESTING

5.1 Classes of Tests

5.2 Expected Software Response

5.3 Performance Bounds

5.4 Identification of Critical Components

5.5 Gameplay Balancing

6. APPENDIX

6.1 Requirements Traceability Matrix

6.2 Target User

6.3 Acknowledgements

1. PREAMBLE

1.1 GAME OVERVIEW

Riot is a reverse tower defence game set in a prison environment. The user is tasked with sending waves of inmates through a gauntlet of guards and obstacles with the goal of breaking out as many inmates as possible.

Riot is broken up into levels, each of which are a different detainment facilities. These facilities will progress in difficulty, starting with classroom detention all the way to a supermax prison. As players progress through the game and successfully lead their escapes they will earn 'rep', a currency in the game which can be used to recruit other inmates.

The game is roguelike by design if they should fail they will begin from the first level. The user is able to end their game session and resume where they left off, however.

*Careful - I don't want their eyes
people recycling from prison
rogue state course and will
check with Mark
to make sure
this isn't what
you're doing*

1.2 VISION STATEMENT

Riot brings to the table the common mechanics of a reverse tower defense style game with a prison themed twist to make users interested when they first play it and keep them interested as the game progresses.

1.3 MAJOR CONSTRAINTS, LIMITATIONS AND RESTRICTIONS

add introducing sentence
Some major constraints helped narrow down and simplify how the software will be built. Other constraints limited some features and functionalities. Aesthetically the software was limited from successful demonstration of what the game could really look like and bring its graphics to potential. These constraints include:

ncurses: The software will be built using the ncurses library. This library allows the software to be displayed as a text-based user interface in an environment similar to the terminal window. Not only is this alone a constraint, but the ncurses library limits the software to what it is going to look like in terms of graphics. As a result the software will only use text-based graphics to portray the game on the screen.

C Language: The software will be written in C to create the game.

User Input: Input will only be taken from the keyboard to interact with the software.

Time Constraint: Due to the fact that the date in which the software needs to be implemented and finished is not far off from now, the software's features have been limited to what is actually feasible given the amount of time to have the software completed and reach all the requirements.

2. GAMEPLAY

2.1 PLAYER OBJECTIVE

On each level, the player's objective is to accumulate as much 'panic' as possible in order to successfully start a riot in order to lead an escape out of the correctional facility. In order to accomplish this objective they must strategically deploy inmates based upon the types of guards the game's landscape.

2.2 CONTROLS

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.

2.3 PROGRESSION

When the user starts the game, the difficulty of the game will be easy and will slowly increase every level you complete. The components of the game will consist of the maps, inmates, and towers.

2.4 GUARDS (TOWERS)


The towers will be placed in different locations every level with different difficulties and roles assigned to them. In the beginning, the classroom, the towers will be considered as teachers, principals,

organization

Keys limited
constant - "panic"?
"guard"?

librarian, gym teacher and in the prison, they will be considered as police officers, the warden, lawyer, attorney, judge. With each having a different role in stopping the prisoners. The roles will change as the game progresses.

Guards

Name	Symbol	Damage	Range	Speed	Special Ability
Crooked	(C)	5	1	3	Only tower that can hit ghost
Doctor	(D)	2	3	1	Slows each unit it hits by half its speed
Warden	(W)	Infinite 	1	1	Calls upon the gods to immediately destroy an inmate <i>summon if place the inmate</i>
Hypnotist	(H)	0	3	2	Puts an inmate to sleep for 2 seconds.
Bob	(B)	3	3	4	None
Sharpshooter	(A)	3	6	3	None
Detonator	(D)	4	5	2	Attacks have area of effect: All enemies in range take damage

good table

2.4.1 PROPERTIES

2.4.2 ATTACKING

2.5 INMATES (PLAYER UNITS)

defined

replay or replace completely

The inmates, controlled by you, will all have different skills. As you go further in the game, you will begin with a larger amount of 'rep', which will allow you to unlock other inmates. For every inmate that escapes, certain amount of credit will be given. The inmates included are your homeboys, the junkie, the gangster, the attorney, soldier, chaplain, the psycho, and the killers. Each having a specific role that can be used to strategize the way you play.

lots of items are being used either inconsistently or without definitions

Name	HP	Speed	Special Characteristic	Price (rep)
Psycho	30	3	Destroys any unit it runs into	20
Homeboy	20	4	None	5
Speedy	10	6	If speedy runs into a unit that unit's speed is increased by 2. if another unit is in front of that unit the effect is negated	40
Fatty	50	1	Reduces all damage taken by half	30

interest

Pretty Lady	20	3	Any guard in her radius will target her.	25
Chaplain	20	4	Unit directly behind Chaplain is invincible for the duration of chaplain's life	60
Ghost	10	4	Immune to all guards except ghostbuster	100
Doctor	15	5	Heals unit directly in front of it for its full	45
Gangster	20	5	All damage taken is reduced by half	40
Time Bomb	1	4	When hp reaches 0, destroys everything in a radius of 3 units (includes both towers and inmates)	200

2.5.1 PROPERTIES

2.5.2 MOVEMENT

defeat

2.6 ENVIRONMENTS

better worded

The maps change every level as you go further on in the game, getting bigger and harder to challenge the player. The maps ~~will~~ consist of different detainment facilities, from a classroom to a supermax prison.

*at this point you are repeating
order → start explaining more
explicitly given section*

2.6.1 LEVELS

In the initial release of the game there are [X] levels. As stated previously, level layouts are stored in external files and can be added or extended by either the user or developer at a later date.

Each map will consist of 1 path represented by the period (.) symbol. The starting point of each path is denoted by the asterisk symbol (*). The end of the path is represented by the dollar sign (\$). Guards denoted by uppercase letters in brackets. Guards are placed along the sides of the path. (could be explained in more detail)

2.6.1.1 THE CLASSROOM (TUTORIAL)

2.6.1.2 JUVINILE DETENTION

2.6.1.3 DRUNK TANK

2.6.1.4 MINIMUM SECURITY FACILITY

2.6.1.5 TIJIUANA LOCKUP

2.6.1.6 MEDIUM SECURITY FACILITY

2.6.1.7 SUPERMAX

2.6.1.8 GUANTANIMO

not stated
could be
taken in
some different
at this
point
(can it be reform
in appendix)

2.6.1.9 LUNARMAX (SPACE JAIL)

2.6.2 PATH

Each map will consist of 1 path represented by the ASCII period symbol(.) . The starting point of each path is denoted by the ASCII dollar sign symbol (\$). The end of the path is represented by an ASCII asterisk(*). Guards are placed in proximity to the path.

*example to
illustrate?*

2.6.3 OBSTACLES

As the game progresses, different sections of the path may have different traits, which may provide an additional challenge to the player.

3. USER INTERFACE

3.1 OVERVIEW

why this size?

The program can be executed and ran within a terminal window with a minimum dimension 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 best user experience. Terminals which do not support UTF-8 characters may default to backup ASCII character sets, however this is platform dependant.

3.2 DESIGN RULES

The two main focuses on the game's design are consistent presentation and intuitive controls.

Both the main menu and the game screen are required to fix inside the 80x24 window and are to be outlined with a border. Every screen is required to prominently display the game's title in order to enforce the game's brand. The game will be consistently controlled using the lowercase alpha keys, and each permissible keypress is to be explicitly indicated with square brackets on the user's screen.

3.3 FRONT END

3.3.1 MAIN MENU

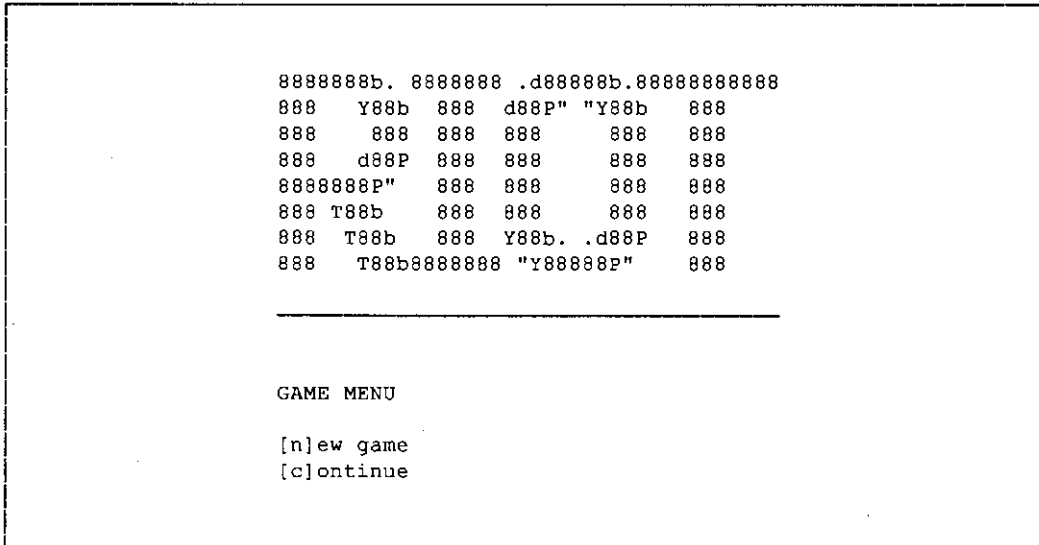


Fig. [x], a mockup of the the game's game menu screen.

3.3.2 NEW GAME

New game will start the player at the first level loaded from disk. It will also overwrite temp.save, a temporary save file, located in /assets.

3.3.3 CONTINUE

← Not sure if the name specified

Continue will load the temp.save file if it exists and resume from the player's last level, else notify the user that it cannot be found and begin a new game.

3.4 MAIN GAME

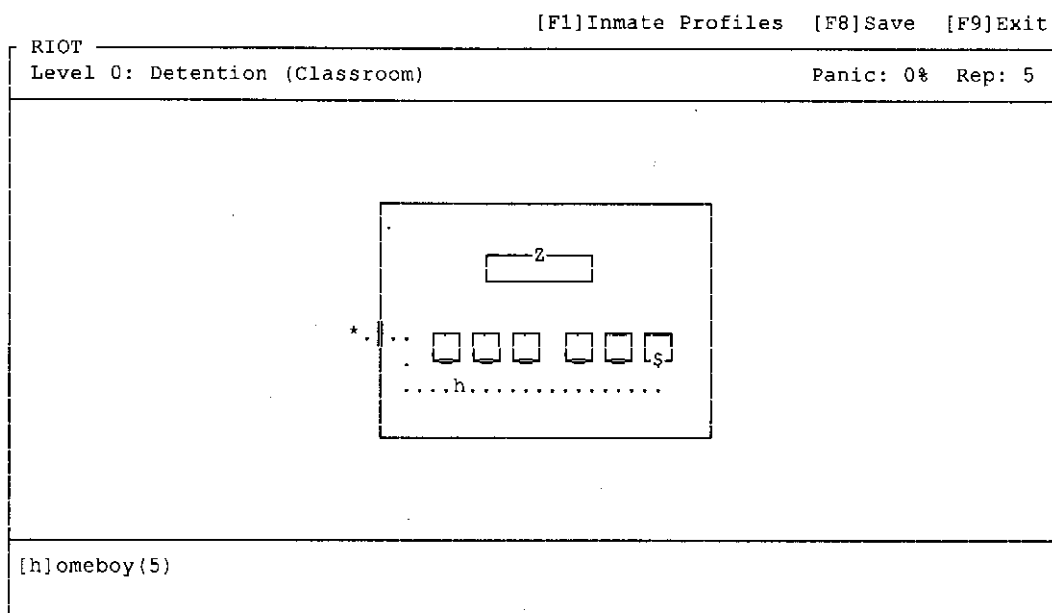
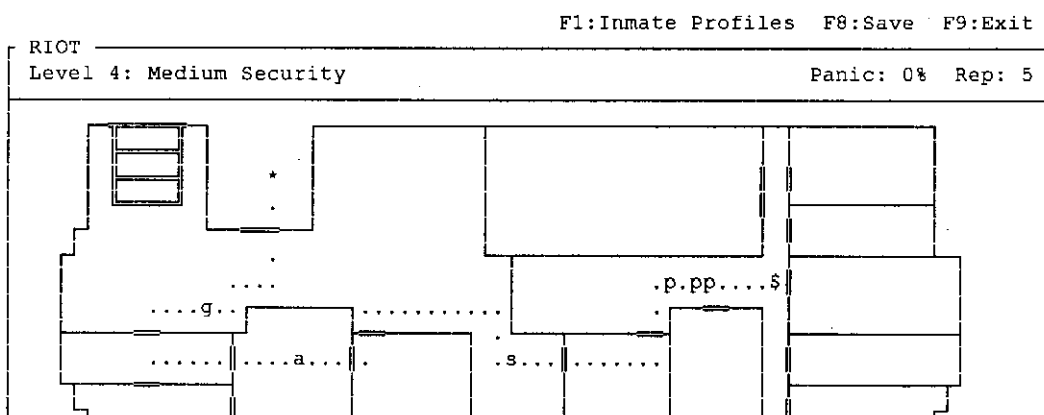


Fig. [x], a mockup of the the game's tutorial level.



recycling
rogue?

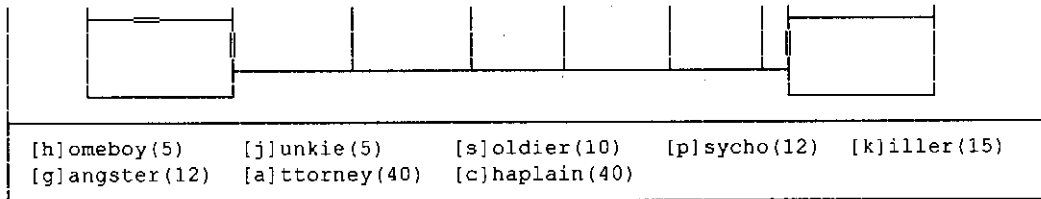


Fig. [x], a mockup of a prison level.

4. SOFTWARE DESIGN

4.1 ARCHITECTURAL 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, ease of development, and to satisfy the game's requirements.

← This is an old plan to specify
The game source files are stored in 'src', header files in 'include', documents in 'docs', executables in 'bin', and asset files (ie. maps and game saves) in 'assets'.

Game logic is divided into four categories: Input/Output, units, the map, and the user interface, each of which are mapped onto source code and header file pairs. These are riotIO, riotUnits, riotMap, and riotUI, respectively.

- in v1.37 when looking for a decision after
riotIO will provide an interface to fileIO which will be used for writing and reading save files on the disk. It will also read map files from the disk and store an internal representation of them in memory as needed. Lastly, it will process user input from the keyboard, interpret it, and relay it to other program subroutines in a meaningful way.

riotUI will provide an interface to draw the to the 'menu', 'header', and 'footer' nCurses windows.

riotUnits will define the characteristics of both inmate and guard units. It will process these unit's actions and manage the interactions between them.

riotMap will interpret the state of the gameboard and units and display them on the 'main' nCurses window.

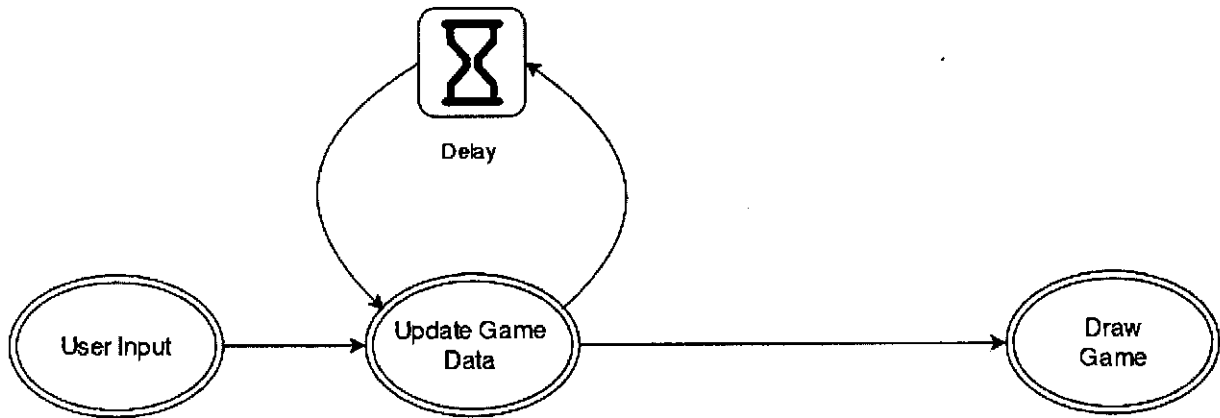
4.1.1 PROGRAM STRUCTURE

```
.
├─ Makefile
├─ README -> ./docs/readme.txt
├─ assets
│   ├── 0.riot
│   ├── 1.riot
│   ├── 2.riot
│   ├── 3.riot
│   ├── 4.riot
│   └─ temp.save
├─ bin
│   └─ riot
├─ docs
│   ├── design.pdf
│   ├── readme.txt
│   └─ testing.txt
├─ include
│   ├── riotIO.h
│   ├── riotMap.h
│   ├── riotUI.h
│   └─ riotUnits.h
└─ src
    ├── riotExec.c
    ├── riotIO.c
    ├── riotMap.c
    ├── riotUI.c
    └─ riotUnits.c
```

*This is for development
or deployment?*

4.1.2 ARCHITECTURE DIAGRAM

4.1.2.1 Model View Control Diagram

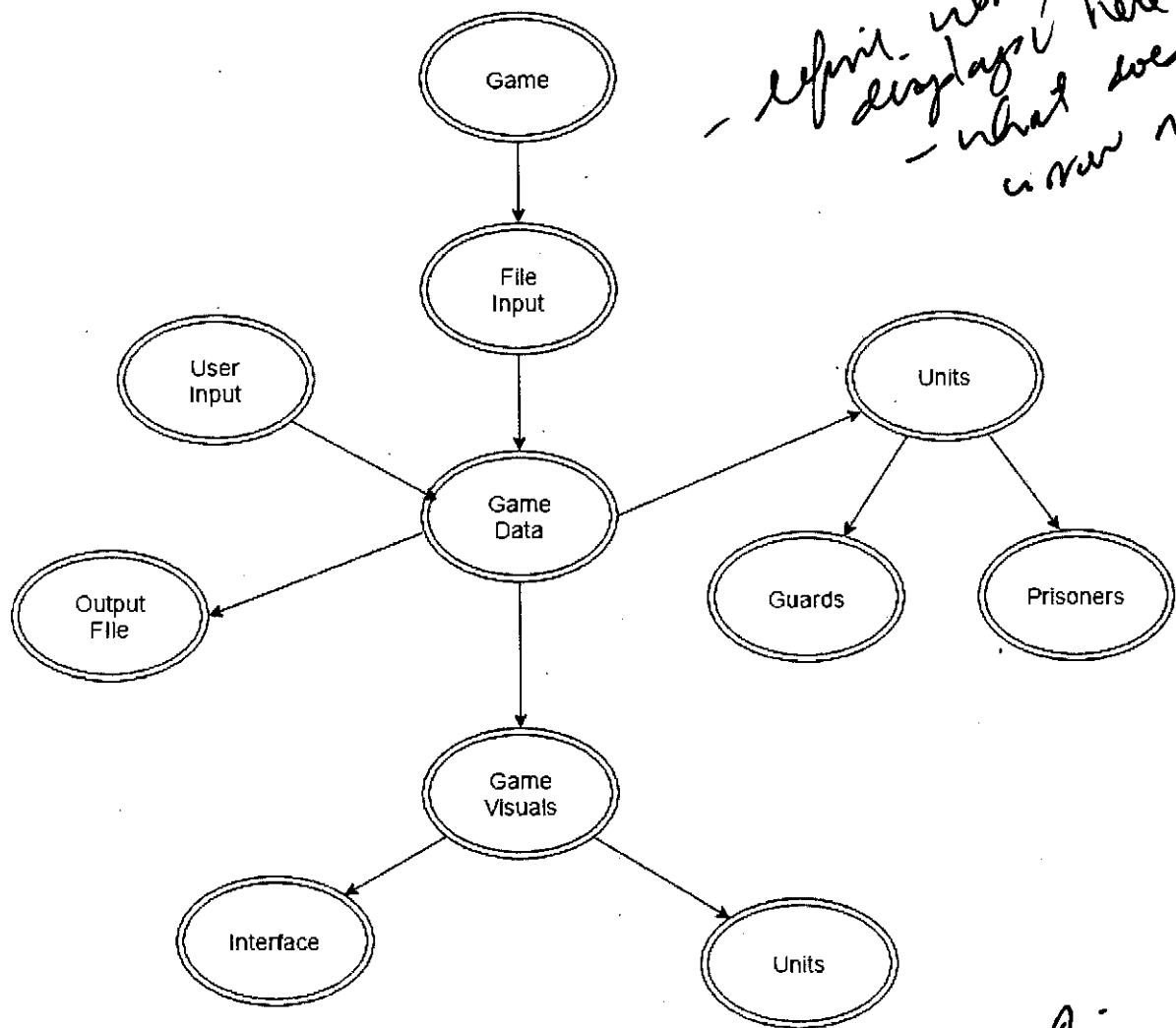


The model view diagram shows how the logic/data processing part of the game will be decoupled from the drawing portion of the game as well as the user input.

In the diagram above, the component "Update Game Data" which could also be referred to as the update loop is where all the data or logic processing takes place. This update loop is continuously called every X amount of time, where X could be any unit of time small enough to provide smooth gameplay without any sort of lag or delay. After the delay is over and the game data is processed the game data is sent to the draw game component where the game is drawn using the ncurses library.

The user input component communicates to the update game data component all the input provided by the user. The update game data component receives the input and processes it once the delay is over, after which the game is drawn again.

4.1.2.2 Game Component 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

4.2 DATA DESIGN

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

4.2.3 STORAGE STRUCTURE

Map files will be parsed and loaded from the disk and then processed into the appropriate internal data structures. While this is less efficient than having had hard-coded them into the software, this allows for levels to be added, patched, or expanded without recompilation.

- hand?

11/1/78

By default these asset files are located within '../assets', relative to the binary file. Alternate map source directories can be used by passing the directory name as a command line parameter to the 'riot' executable. Map files must have a '.riot' extension to be recognized and parsed in lexicographical order.

organization need
for some of these
hard disk structure

This is higher
 pursuing more
 when from
 looking
 what if
 Mrs.
 needs
 extra

Mr. Mrs. Wells
Hester

Mr. Mrs. Wells
Hester

Mr. Mrs. Wells
Hester

should be
of damage
ted by se
its to th
it rende

ting will be done

Wong, A. T. H.

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 don't have a sufficient amount of funds for that specific unit. Then also iterating between the store and the game scene seeing if there are any problems.

Unit testing will also need to be conducted on what happens when a unit die, will it still get targeted by the NPC's or prevent other units from passing by it. This test is conducted to also see if the game will end properly if there are no more available units to summon.

5.2 EXPECTED SOFTWARE RESPONSE

5.3 IDENTIFICATION OF 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.

5.4 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 blackbox testing will help test the difficulty and see if it is too hard for newer players.

6. APPENDIX

6.1 REQUIREMENTS TRACEABILITY MATRIX

6.2 LIMITATIONS

6.3 TARGET USER

6.4 ACKNOWLEDGEMENTS