|  |
| --- |
|  |
| Cut the Power |
| A Data Communications Project |
|  |
| **Set 4O** |
| **April 8th, 2014** |

|  |
| --- |
|  |

Table of Contents

[Abstract 4](#_Toc384669630)

[Synopsis 5](#_Toc384669631)

[Controls 6](#_Toc384669632)

[Features 7](#_Toc384669633)

[Input: 7](#_Toc384669634)

[Gameplay: 7](#_Toc384669635)

[Graphics: 8](#_Toc384669636)

[Client: 8](#_Toc384669637)

[Server: 8](#_Toc384669638)

[Sound: 9](#_Toc384669639)

[Other Features: 9](#_Toc384669640)

[Development Stages: 10](#_Toc384669641)

[Milestone 1: 10](#_Toc384669642)

[Milestone 2: 10](#_Toc384669643)

[Milestone 3: 10](#_Toc384669644)

[Gantt Chart: 11](#_Toc384669645)

[Teams: 12](#_Toc384669646)

[Project Leads: 12](#_Toc384669647)

[Team Leads: 12](#_Toc384669648)

[Developers: 12](#_Toc384669649)

[Network: Client & Server: 12](#_Toc384669650)

[Graphics: 12](#_Toc384669651)

[Input: 12](#_Toc384669652)

[Game Logic: 12](#_Toc384669653)

[Gameplay: 13](#_Toc384669654)

[State Diagram for the Game: 14](#_Toc384669655)

[Input: 15](#_Toc384669656)

[Milestones: 15](#_Toc384669657)

[Milestone 1 – Base Systems and Components 15](#_Toc384669658)

[Milestone 2 – Menu 15](#_Toc384669659)

[Milestone 3 – Mouse Look Control and Key Mapping 15](#_Toc384669660)

[State Diagram: 16](#_Toc384669661)

[Key Input System: 16](#_Toc384669662)

[Pseudocode: 17](#_Toc384669663)

[Game Logic: 18](#_Toc384669664)

[Milestones: 18](#_Toc384669665)

[Milestone 1 – Base Systems and Components 18](#_Toc384669666)

[Milestone 2 – Extending the current systems and victory conditions 18](#_Toc384669667)

[Milestone 3 – extending the gameplay mechanics 18](#_Toc384669668)

[State Diagrams: 19](#_Toc384669669)

[Game Loop: 19](#_Toc384669670)

[Movement and Collision System: 20](#_Toc384669671)

[Pseudocode: 21](#_Toc384669672)

[Network: Client: 24](#_Toc384669673)

[Milestones: 24](#_Toc384669674)

[Milestone 1 – Basic Framework 24](#_Toc384669675)

[Milestone 2 – Floor Information and Lobby 24](#_Toc384669676)

[Milestone 3 – Chat System, Polish and Additional Features 24](#_Toc384669677)

[State Diagram: 25](#_Toc384669678)

[Network Router Thread: 25](#_Toc384669679)

[TX Thread: 26](#_Toc384669680)

[RX Thread: 26](#_Toc384669681)

[Pseudocode: 27](#_Toc384669682)

[Milestones: 29](#_Toc384669683)

[Milestone 1 – Base System 29](#_Toc384669684)

[Milestone 2 – Floors and Lobby Systems 29](#_Toc384669685)

[Milestone 3 – Features and additional Systems 29](#_Toc384669686)

[State Diagrams: 30](#_Toc384669687)

[Server Process Flow: 30](#_Toc384669688)

[General Controller: 31](#_Toc384669689)

[Inbound Switchboard: 32](#_Toc384669690)

[Connection Manager: 33](#_Toc384669691)

[Outbound Switchboard: 34](#_Toc384669692)

[Gameplay Controller: 34](#_Toc384669693)

[Pseudocode: 35](#_Toc384669694)

[Graphics: 37](#_Toc384669695)

[Milestones: 37](#_Toc384669696)

[Milestone 1 – Simple Map 37](#_Toc384669697)

[Milestone 2 – Better Maps/Menu 37](#_Toc384669698)

[Milestone 3 –Fog of War 38](#_Toc384669699)

[State Diagram: 39](#_Toc384669700)

[Graphics Renderer: 39](#_Toc384669701)

[Pseudocode: 40](#_Toc384669702)

[Sound: 41](#_Toc384669703)

[Milestones: 41](#_Toc384669704)

[Milestone 1 – Play Background Sound 41](#_Toc384669705)

[Milestone 2 – Play sounds specific to an item/player 41](#_Toc384669706)

[State Diagram: 41](#_Toc384669707)

[Pseudocode: 41](#_Toc384669708)

[Coding Style/Choices: 42](#_Toc384669709)

[Github Location: 42](#_Toc384669710)

[Screenshots: 43](#_Toc384669711)

[Logo screen: 43](#_Toc384669712)

[Main Menu: 43](#_Toc384669713)

[Options: 44](#_Toc384669714)

[Credits: 44](#_Toc384669715)

[Selection: 45](#_Toc384669716)

[Server IP Address/Username: 45](#_Toc384669717)

[Driver Animation: 46](#_Toc384669718)

[City Animation: 46](#_Toc384669719)

[Lobby: 47](#_Toc384669720)

[Floors: 47](#_Toc384669721)

[Parking Lot: 47](#_Toc384669722)

[Floor 1: 48](#_Toc384669723)

[Floor 2: 48](#_Toc384669724)

[Floor 3: 49](#_Toc384669725)

[Floor 4: 49](#_Toc384669726)

[Floor 5: 50](#_Toc384669727)

[Floor 6: 50](#_Toc384669728)

[Floor 7: 51](#_Toc384669729)

[Floor 8: 51](#_Toc384669730)

Abstract

The game is a Rogue-like multiplayer game built for the Linux system. The game will allow multiple users to join two teams, where, there will be collision detection and a point system. The users will be controlled through the WASD or ARROW keys. There is a fog of war implemented that prevents users from seeing each other. The game library of choice is the SDL2 library. Graphics are made in class, and sound is a mix of custom music and music from the internet. Users will join a server to a maximum of 32 players.

# Synopsis

Information Decryption and Discovery Bureau (IDB Inc.) has been hired by the government to spy on its citizens. They have been placing legalized viruses throughout the country and its global partners to spy on these citizens.

You are a concerned Citizen and you are sick of the gloat of legal viruses being deployed throughout your country and the invasion of privacy. You want the truth to come out and you want the “people” to know what type of things the government has been doing. You are not alone and will be joined in this fight with fellow concerned citizens.

You are a simple Security Guard, outsourced by Innovative Data Brokerage. You don't care about what they do or how they do it. Your only concern is when you will get your paycheck. You have bills and you have people that depend on you. You and your fellow employees have one task; ensure IDB Inc. property is safe.

What side will you play?

# Controls

Cut the Power controls only support keyboard use. The default controls for the game are:

* UP = W (Customizable)
* DOWN = S (Customizable)
* LEFT = A (Customizable)
* RIGHT = D (Customizable)
* ACTION = SPACE (Customizable)
* OPTION MENU = ESC
* SPECIAL TILES = E
* CHAT = ENTER

The game supports key binding and is customizable by having the user go through the options menu. The game will support the following button keys: A-Z & 0-9.

The game menu and player selection is controlled by mouse input

# Features

The following are notable features surrounding Cut the Power. These features are not specific to any groups contribution, but to the specific part of the game that it impacts.

## Input:

These are the features that are specific to Input:

* Key Binding: Users are capable of changing the controls for the player. The user needs to go to the Options->Key Map and enter the corresponding keys they would like to replace. The keys that are changeable are from A-Z and 0-9.
* True Type Font: The game uses the PlayerSelect2.ttf true type font for all text representations.
* Window Resizing: The window is resizable and allows the game to be played on multiple resolution screens.
* Full Screen Mode: The game is interchangeable between full screen and window screen mode.
* Chat: Client/Server chat is displayed to users in the game, depending on the team they are placed on. The messages will be in a specific colour depending on who speaks (red for yourself, white for others and yellow for the Server)
* Message Displays: Messaging fades out after a brief period of time.
* Scaling: The game is scaled accordingly. If the window is wider than tall, black bars are created and graphics are scaled accordingly.
* Option menu: If the user presses ESC on the keyboard, a menu is displayed; allowing the user to turn the sound on/off, full screen on/off and either exit the options menu or exit the game.

## Gameplay:

These are the features that are specific to Gameplay:

* Object Interaction: Players are able to interact with an object by going to the object and pressing the ACTION key on the keyboard. This interaction will change a state of an object.
* Stair interaction: Interaction with the stairs allows the players to switch between floors. When a player collides with a stair, they will be transported to another floor.
* Player/Object Collisions: Players can collide with each other and depending on the game feature affect the game play. Players on the same team can obstruct each other. Players on opposite teams use the player collision for an action.
* Custom tiles: Custom tile interaction has been added and can be randomly placed on the map.
* Acceleration/De-acceleration: Able to change the speed of the player (either directly or through a “tile”).
* Secondary Skins: Implemented a system that selects a custom secondary skin for players. This system is random for each player and has a percentage of approx. 20% of occurring along with a custom sound that plays.
* Map editor: A custom map editor was written with JQuery to use along with the Tiled program that was used with creating maps.
* Chat: Integration with input to support a chat system across the game. The chat system is able to happen mid-game
* FPS: Frame per second scaling to set frames per second at 30 fps. This system scales players either higher or lower to ensure that all player movements are smooth.
* Special Tiles: Special tiles are placed when the user presses E on the keyboard. The types include a forward movement, backward movement, slow down and speed up. The time of the effect is 5 seconds and activation of these types of tiles is when the user goes over a special tile power up found on the map.

## Graphics:

These are the features that are specific to Graphics:

* Maps: Created through a tile sheet program. First implemented with Tiled, then with the custom editor. There is a total of 9 floors for gameplay and a lobby floor.
* Objects: Separate entities/objects that can be placed randomly on the map of variable sizes
* Fog Of War: A fog of war for players that prevent users from seeing anything beyond their positioning. All objects must be searched on the map to be found. Initially, the map will be black to the robbers, but as they explore, they will get a gray overlay which represents the travelled portions. Cops will begin with this format.
* Custom player selection: After the user presses Play, the user will be shown a character select screen that will display the students and instructor in the class. The user will then be able to choose a specific player or click random to have a random character chosen.
* Custom players: Each student in the class and the instructor has a custom 8 bit character that resembles them.
* Cut scenes: There is a load and intro screen for the game.
* Custom Animations: Load screen/intro screen are pieced together as multiple images and put together using the animation system.
* Special tiles: Special tiles have a custom animation.
* Custom fonts: Text is drawn dynamically on the screen.

## Client:

These are the features that are specific to Client:

* Special Tiles: Special tiles are place-able on the map. They have a timer of 5 seconds. Built on a system created by Graphics.
* Map Reset: Map reset for stair changes.
* Fog of War reset: The fog of war is reset.
* Quit/Rejoin: Balances the quit/rejoin so that the game doesn’t crash when players join, exit or rejoin.
* Lobby waiting: Lobby waiting occurs while players are still playing the game. Sides are selected during this time.
* Packet optimization: Optimized the overall size of the packets, decreasing the size of the packets and increasing efficiency.
* Thread/IPC model: The client portion of the game was created with a thread/IPC model.
* Packet Balancing: The packets are balanced to ensure flow and efficiency.

## Server:

These are the features that are specific to Server:

* Detailed Logging: Detailed message logging is done on the server. Allows the server to keep track of current results in the game.
* Disconnects: The server handles disconnects mid-game and allows current players to finish the game without distraction.
* Late Joins: If a player joins late to the game, they will wait in the lobby till a new round occurs.
* Minimized Packet Size: Packets are designed to be the smallest size possible, and accordingly are optimized­­­ as well.
* Packet Injection: Able to simulate any packet that would have been sent by the Client, allowing features to be tested.
* Keep alive implementation: Check to see if the client sent a packet.
* Combination Select/Threading model: The server was designed with a combination of Select and Threading to support the various packets/connections.
* Team Determination: The server balances the team, ensuring that teams are fair. If not, then the favour will go to the Guards with one additional member on that team.
* Server Chat/Messaging: The server implements the chat into the game, allowing the server to message the clients. Examples would be notifying win situations, countdown and captures.
* Chat “encryption”: All messages from the robbers will be “encrypted” to the cops and as time progresses, the messages will begin to decrypt.
* Lobby waiting: Lobby waiting occurs while players are still playing the game. Sides are selected during this time.
* Down speed: The average speed is

## Sound:

These are the features that are specific to Sound:

* Sound Effects: Play according to actions such as a custom animation
* Sound Files: Sounds change throughout the game and are timed to a state change (Load to Title to Game)
* Fog of War signal: When a player has entered the fog of war, a sound is played.
* Custom Player sound effect: When a special player is selected, a sound plays.

## Other Features:

These are the features that are not specific to any group:

* WAN: Playable over WAN. Test server IP is: 216.232.166.239.
* Custom Packaging: Packaging was created for the game
* Custom Manual: A custom game manual was created for the game
* Trailer: A game trailer was created for the game.

# Development Stages:

Development of the game has been broken into three major milestones.

## Milestone 1:

Due by **February 21st**. The gameplay for the first milestone is a very simple implementation of Cops vs. Robbers. The game will allow 2 users to connect to each other through a separate server. Each user will connect to this server and when both users have connected, the game will commence. Upon opening, the game will display a map with an external wall preventing users from moving off-screen and internal walls to impede their movement inside the map. Throughout the map, there will be 4 “systems” for the Robber to compromise. To win, the Robber must simply go over three of the four systems and the Cop must collide with a Robber. To compromise a system, a Robber must simply go over the system. The Cop must collide with a Robber to arrest them. The game will consist of seven rounds. A round will end when either the Cop arrests the Robber or the Robber compromises three objectives.

## Milestone 2:

Due by **March 7th**. Milestone 2 focuses on gameplay robustness and increasing the player count. The game will allow two teams of up to 16 players each to participate in a game (for up to 32 players). Multiple floors will be introduced, which will be accessed through stairs. Players will use stairs to navigate the floor system introduced in this milestone (consisting of up to 8 floors. Randomized “objectives” will be placed across the floors. The total floors available will be determined by the number of players connected, with every 4 players generating a new floor. After pressing Start on the menu screen, users will be taken to a lobby, where they will see the other users connected within the game. The users will be able to select which team they would like to play for; however, the game will rectify any balancing issues.

## Milestone 3:

Due by **March 20th**. The third and largest milestone will introduce many of the games crucial features. The lobby will now be a “playable” map, where users will be able to decide by their location on the map to the team for which they would like to play. This lobby will be much like a floor in the game and will be interactive. Users will be able to explore the map in new ways such as secret passages and ventilation systems. User interaction with the objectives will now be possible.

# Gantt Chart:



# **Teams**:

## Project Leads:

|  |  |
| --- | --- |
| Jordan Marling: | Mateusz Siwoski: |

## Team Leads:

|  |  |  |
| --- | --- | --- |
| Andrew Burian | Cory Thomas | Konstantin Boyarinov |
| Shane Spoor | Josh Campbell | Robin Hsieh |

## Developers:

|  |  |  |  |
| --- | --- | --- | --- |
| Damien Sathanielle | Vincent Lau | Tim Kim | Chris Holisky |
| Sam Youssef | Ramzi Chennafi | Ian Davidson | Clark Allenby |
| German Villarreal | | Abhishek Bhardiwaj | |

## Network: Client & Server:

|  |  |  |
| --- | --- | --- |
| Andrew Burian | Chris Holisky | German Villarreal |
| Shane Spoor | Ramzi Chennafi | Abhishek Bhardiwaj |

## Graphics:

|  |  |  |
| --- | --- | --- |
| Robin Hsieh | Sam Youssef | Tim Kim |
| Konstantin Boyarinov | Damien Sathanieele | Mateusz Siwoski |

## Input:

|  |  |  |
| --- | --- | --- |
| Vincent Lau | Jordan Marling | Cory Thomas |

## Game Logic:

|  |  |  |
| --- | --- | --- |
| Ian Davidson | Clark Allenby | Josh Campbell |

# Gameplay:

The game is top-down, rogue-like and multiplayer-centric. The game is employs a points based system where each successful compromise is a point for the “hackers” and each arrest is a point for the “Security Guard”. Individual players will also compete for the highest number of points on their team each round. The game will be round based with rounds being set by the users upon connecting to the lobby. Once a game has ended, users will be taken back to the lobby where they will be able to commence another match.

When a game begins, there will be a number of floors available (based on the number of players) to a maximum of 8 floors. The floors will be connected by stairs which users will “enter” to change floors. The players will be randomly placed around floors and objectives will be randomly placed throughout floors. Users will also be able to use randomly spaced tools.

The game will allow for a maximum of 32 players with teams having up to 16 players each.

# State Diagram for the Game:



# Input:

## Milestones:

### Milestone 1 – Base Systems and Components

Due by **February 21, 2014**. This milestone will include 2 systems: a basic key input system and a mouse input system that updates the “Key Command” component and “Mouse” component for the player and menu entities. We will work on the setup menu from this stage onward and should go to the end of Milestone 2.

### Milestone 2 – Menu

Due by **March 6, 2014.** During this stage, we will be finishing the initial setup menu with textbox inputs and buttons for modifiable connection parameters. After finishing that, we are helping out the Gameplay group, dedicating more resources to the gameplay systems.

### Milestone 3 – Mouse Look Control and Key Mapping

Due by **March 20, 2014.** For this final milestone, we will be working side-by-side with the Gameplay group to implement mouse looking and their final goals. On our own extra time (if applicable), we will be implementing a key mapping feature, using file parsing to attach keys to commands as opposed to hard-coding the relationship.

## State Diagram:

### Key Input System:



## Pseudocode:

KeyInputSystem

{

if (SDL\_QUIT is triggered)

close program

set previous state of the keyboard to the current state

update the current state of keyboard

If a text field is focused

add alphabetical keys pressed to the text value of the text field

//end if

else

while looping through keys pressed

if a command can be ascertained

update command

//end if

//end while

//end else

}

MouseInputSystem

{

get new x and y positions

set previous click states to current states

while looping through entities

update all entities with mouse components

if entity is a text field

if lclick previous state is down and current state is up AND click happens in text field area

set focus

loop through other text fields and remove focus

//end if

//end if

if entity is a button

if lclick previous state is down and current state is up AND click happens in button area

set command component for button

//end if

//end if

//end while

}

# Game Logic:

## Milestones:

### Milestone 1 – Base Systems and Components

Due by **February 21, 2014.** Includes a basic map/player collision system, a player/player collision system, a basic movement system that simply moves the player to a new x and y coordinate based on input coming in, and designing the components needed to create these systems.

### Milestone 2 – Extending the current systems and victory conditions

Due by **March 6, 2014.** This milestone will include creating randomized targets for hackers, a points/round system in order to create victory conditions, a system for handling victory conditions.

### Milestone 3 – extending the gameplay mechanics

Due by **March 20, 2014.** This milestone will include a predictive movement system in order to compensate for any lag that may occur between the client and the server. Other possible systems that may be developed in this milestone include: a basic AI system, traps, changing map elements (doors, pushing desks, cutting the power to a room or turning it back on, etc.).

## State Diagrams:

### Game Loop:



### Movement and Collision System:



## Pseudocode:

Function MovementSystem(World)

find the input entity

//finding the controllable entity + processing

for i = 0 to MAXENTITIES

if world.mask[i] equals controllable entity mask

create temporary entity

assign actual controllable entity values to temporary entity

if input == up

temporary entity's y location -= temporary entity's velocity

end if

if input == down

temporary entity's y location += temporary entity's velocity

end if

if input == left

temporary entity's x location -= temporary entity's velocity

end if

if input == right

temporary entity's x location += temporary entity's velocity

end if

if controllable entity has collision properties

if Collision\_System(World, Temporary Entity) equals no collision

actual entity gets assigned temporary entity's position data

else if collision with stair

change floor

else if collision with guard and actual entity is a hacker

kill hacker

else if collision with hacker and actual entity is a guard

kill hacker

else if collision with target and actual entity is a hacker

destroy target

end if else

else

actual entity gets assigned temporary entity's position data

end if else

end if

end for

end function

Function Collision\_System(World, tempEntity)

if wall\_collision(world, tempEntity) equals true

return Collision with Wall occurred

end if

if stair\_collision(world, tempEntity) equals true

return Collision with Stair occurred

end if

if guard\_collision(world, tempEntity) equals true

return collision with guard occurred

end if

if hacker\_collision(world, tempEntity) equals true

return collision with hacker occurred

end if

if target\_collision(world, tempEntity) equals true

return collision with target occurred

end if

return no collision

End Function

Function wall\_collision(World, tempEntity)

find tempEntity approximate position on the grid

check world.level at tempentity approximate position to see if there is a wall

if wall == true

return collision occurred

else

return no collision

end if

End Function

Function stair\_collision(world, tempEntity)

find tempEntity approximate position on the grid

check world.level at tempentity approximate position to see if there is a stair

if stair == true

return collision occurred

else

return no collision

end if

End Function

Function guard\_collision(world, tempEntity)

loop through all entities in world

if world entity is a guard

check to see if tempentity overlaps with found guard entity

if collision occurred

return collision occurred

end if

end if

end loop

return no collision occurred

End Function

Function hacker\_collision(world, tempEntity)

loop through all entities in world

if world entity is a hacker

check to see if tempentity overlaps with found hacker entity

if collision occurred

return collision occurred

end if

end if

end loop

return no collision occurred

End Function

Function target\_collision(world, tempEntity)

find tempEntity approximate position on the grid

check world.level at tempentity approximate position to see if there is a target

if target == true

return collision occurred

else

return no collision

end if

End Function

# Network: Client:

## Milestones:

### Milestone 1 – Basic Framework

Due by **February 21st**. The first milestone is to have the basic framework for the client-side networking in place. The framework includes our network controller (which routes data between the threads and Gameplay), the send and receive threads, and the API functions for communication with the Gameplay module. The only information passed between the Gameplay, Client Network and Server modules for this first milestone are basic handshake information, position updates, and objective updates.

### Milestone 2 – Floor Information and Lobby

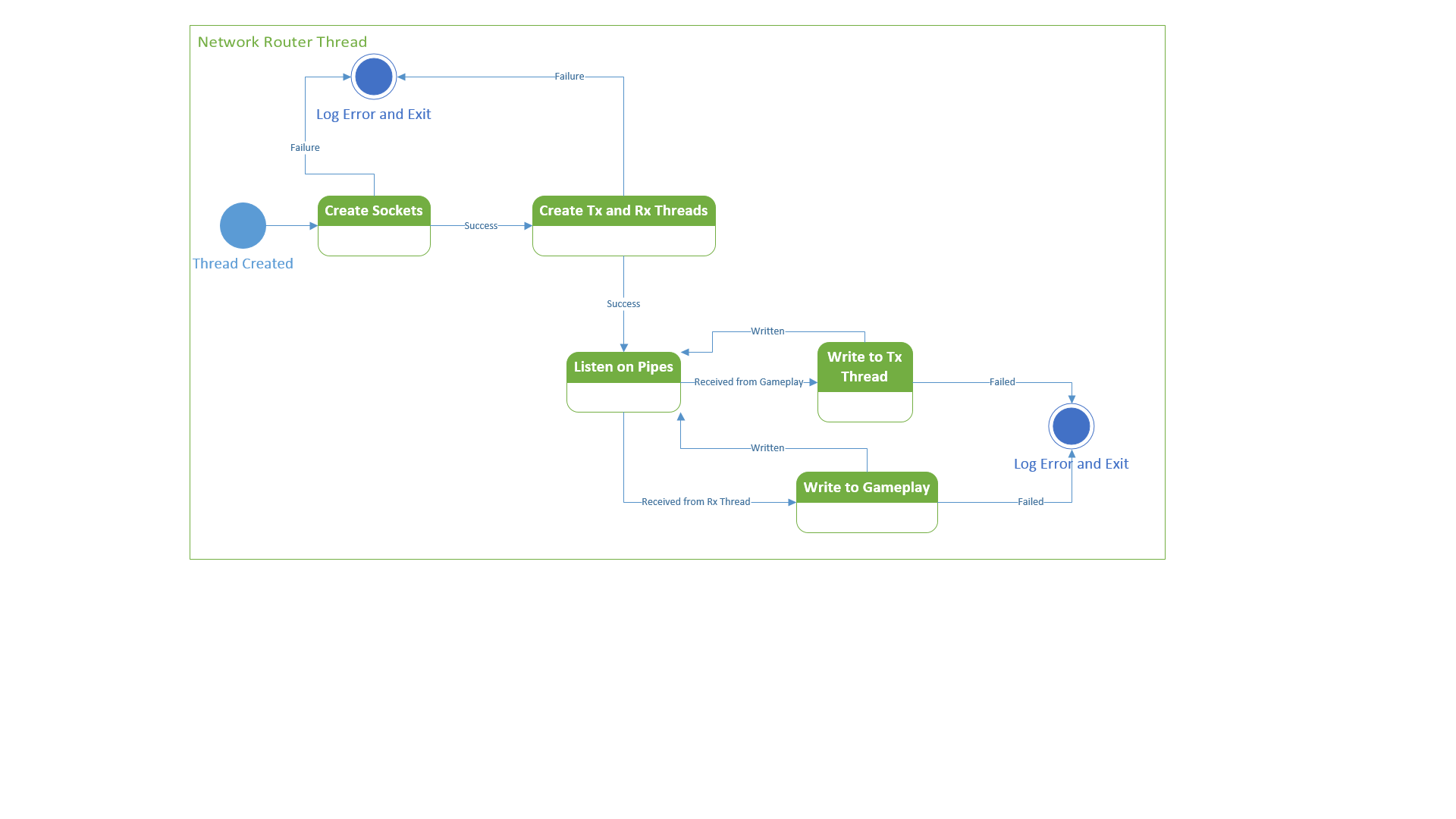
Due by **March 6, 2014.** The second milestone adds floor information and the initial lobby system. The Client Network module will send its player’s name and other player details to the server, and will receive and relay to the Gameplay module the details of everyone else in the lobby. Once the game starts (indicated by the start of game packet from the server), the Client Network module will handle the position updates as before as well as floor change requests (client side) and confirmations (from the server).

### Milestone 3 – Chat System, Polish and Additional Features

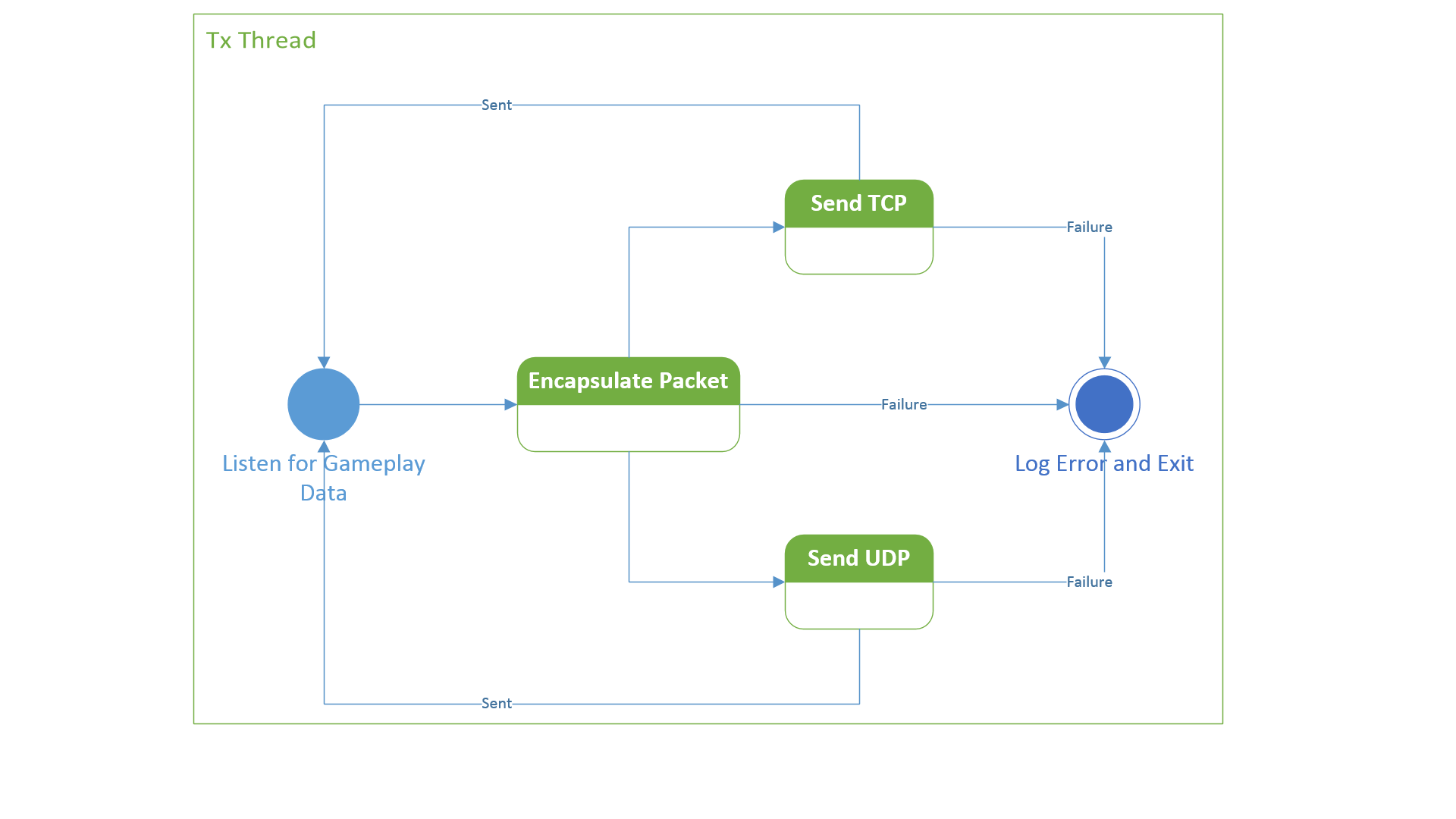
Due by **March 20, 2014.** The final milestone will add support for a chat system between team members. We will also perform some rigorous final testing Depending on time constraints and the states of the other teams, we will either assist other groups or implement additional features (WAN support, cross-compatibility with Windows/Mac, etc.).

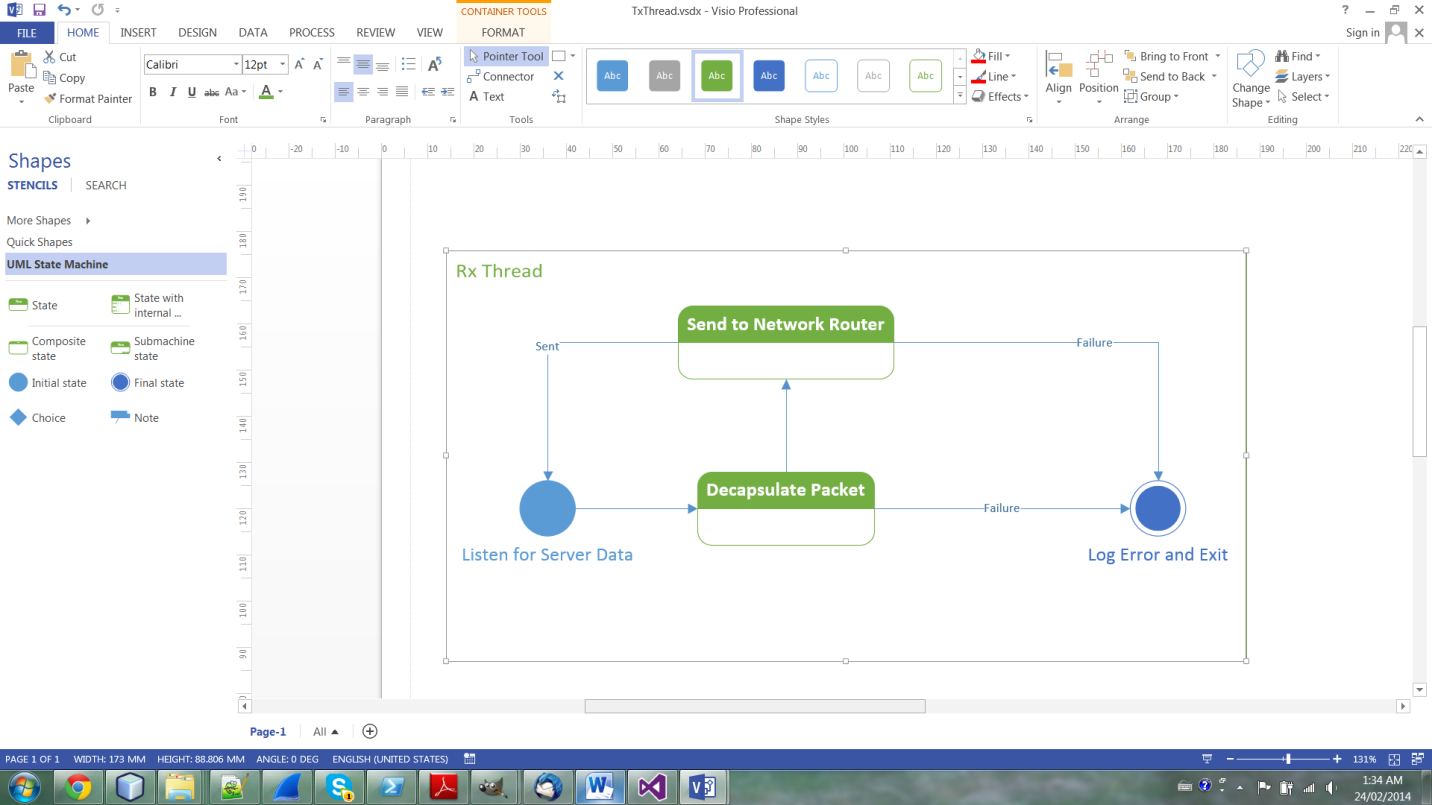
## State Diagram:

### Network Router Thread:



### TX Thread:



RX Thread:

## Pseudocode:

Send Pseudocode{  
While true  
   Wait on pipe for data  
   Read struct from pipe  
   Determine protocol  
   Send data on correct socket  
}

Encapsulate Pseudocode{  
Take data  
Switch on type  
   case type x:  
   Set protocol in packet struct  
   Write struct into packet struct buffer  
Pass struct to network  
}

Receive Pseudocode{  
While true  
   Wait on socket  
   Read the socket  
   Send the data to the pipe  
}

Decapsulate Pseudocode{  
 Read packet type  
 Write packet type to pipe  
 Copy data into struct  
 Send data to network  
}

Network Pseudocode{  
Read gameplay pipe ends from parameters  
Pipe twice  
Start send and receive threads, passing pipes  
While true  
   Select on gameplay and receive pipes  
   If we receive data on gameplay pipe  
   Frame data  
   Write to send pipe  
   If receive data on receive thread pipe  
   Decapsulate data  
   Write to gameplay pipe  
}

Gameplay Write Wrapper Pseudocode{  
 Write size of packet to pipe  
 Write type of information/packet to pipe  
 Write struct to pipe  
}

Gameplay Read Wrapper Pseudocode{  
Read size of packet  
Read packet type and store in int pointer (passed by gameplay)  
Allocate struct for packet  
Read packet into struct  
Return the struct  
Read Gameplay Packet   
Read size of struct to be received  
Read struct into buffer  
Write struct to pipe  
 }Network: Server:

## Milestones:

### Milestone 1 – Base System

Due by **February 21, 2014.** Set up the basic program structure for the server, including all the IPC, and the completed framework for all future feature controllers to be plugged into. Supports up to 2 players, and will support movement and game objectives. Players joining the game will be immediately put on a gameplay area, and the game will start as soon as 2 players have entered the area.

### Milestone 2 – Floors and Lobby Systems

Due by **March 6, 2014.** Add the floor concept to gameplay. Players will first spawn in the “Lobby” floor, where they will be able to choose their teams. Gameplay starts after a set time has elapsed and a minimum number of players are present. Players are then moved to a floor by the server. Players can then move freely between gameplay floors. Possibly also introduce the chat feature.

### Milestone 3 – Features and additional Systems

Due by **March 20, 2014.** Add any additional controllers for systems including the chat, traps, alarms, and any other features.

## State Diagrams:

### Server Process Flow:



### General Controller:



### Inbound Switchboard:



### Connection Manager:



### Outbound Switchboard:



### Gameplay Controller:



## Pseudocode:

UI{

prompt setup info

pass IPC packet 0 to Inbound Switchboard

while running

get input

if quit

create quit packet

send to switchboard

exit

if other

other

}

CONNECTION MANAGER{

create and bind a TCP listen socket

loop

accept new connection

see if there is enough space in the game to accept a new client

if game is full

send pack packet 2 with game connect denied

FIN connection

Close socket

if space is available

assign new client a player number

send packet 2 back to client with connect accepted.

send IPC packet 1 to Inbound Switchboard

}

INBOUND SWITCHBOARD{

allocate socket lists

wait for IPC packet 0 from UI

pass IPC packet 0 to Connection Manager, Outbound Switchboard, Gameplay, and General

loop

listen on all sockets

if connection manager socket

add new connection to socket to list of inbound sockets

create a new UDP socket based on the information

Pass the new info to Outbound Switchboard, Gameplay, and General

if udp socket

receive the size of the largest UDP packet

determine which packet type it is

cast the received data to the appropriate structure

if packet 10 (Movement)

pass to Gameplay

if tcp socket

if socket is closed or terminated

remove the socket and it's UDP counterpart

create IPC packet 2 (Player lost) and pass it to Connection Manager, Outbound Switchboard, General, Gameplay

close descriptors

else

read in the packet type

fill the appropriate structure

if packet 8 (gameplay update)

pass packet to General

}

GAMESTATUS CONTROLLER{

initialize objectives array

wait for IPC packet 0 (setup)

loop

listen on ipc socket

update objectives

send to outbound switchboard

}

GAMEPLAY CONTROLLER{

\*just milestone1\*

wait for IPC packet 0 (setup)

loop

listen on ipc-socket

read player information

update player information

send to outbound switchboard

}

OUTBOUND SWITCHBOARD{

allocate tcp and udp socket list

wait for IPC packet 0 (setup)

loop

listen on socket

read packet type

fill appropriate structure

if IPC packet 1 (New player)

add descriptors to lists

if IPC packet 2 (Lost player)

remove descriptors from list

if Packet 1 - 13

read 32b (4B) more from socket to get send-to-player flags

send received packet to all players specified in flags

}

# Graphics:

## Milestones:

### Milestone 1 – Simple Map

Due by **February 21, 2014**. This milestone will include the game’s initialisation. The first iteration of the game will follow a very simplistic implementation of Cops vs. Robbers. The game will display the map on screen, render up to two players, and allow for user input to maneuver a player around the map. When the game begins, there will be a map that will be displayed with a perimeter wall to prevent users from going off-screen as well as some internal walls to impede players. Throughout the map, there will be 4 placed “objectives” that the Robber must compromise. 

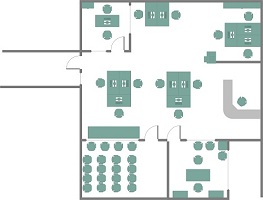


Figure : A segment of a map from a user’s point of view



Figure : Our reference in designing the style of the game

Milestone 2 – Better Maps/Menu  
Due by **March 6, 2014** . The overall graphics and map details will be more polished from the first milestone.

Multiple floors will be introduced with a maximum of 8, which will be accessible through stairs. When a player enters a stairwell going up or down, they will be moved up or down a floor accordingly with the screen going blank The “objectives” from milestone one will now be randomized objects throughout the map.

A pre-game menu screen will also be added which will serve as a more convenient way for users to enter server information with the addition of graphical options. After pressing Start on the menu screen, users will be taken to a lobby, where they will see the other users connected within the game.

### Milestone 3 –Fog of War

Due by **March 20, 2014**. Fog of war will also be added as a feature in this milestone. Players on both sides will have a circle of vision, giving a live view of the players on the map. They will also share vision with teammates on the same floor.

The robbers will start the game with no vision, except for their own, and vision of their teammates. As they explore, they will reveal the map, example follows:



Figure : Fog of War

The cops will start the game with vision of the map layout, as if they have explored the entire map, but not vision of opposing players. As example above, the cops view of the map is a darker shade where the map is revealed.

## State Diagram:

### Graphics Renderer:



## Pseudocode:

int main(){

initialize(){}

create thread for loading images()

bool quit = false;

SDL\_Event e;//Event handler

while( !quit ) //While application is running{

while( SDL\_PollEvent( &e ) != 0 )//Handle events on queue{

if( e.type == SDL\_QUIT ) //User requests quit.{

quit = true;

}

}

render\_system(&world, surface);

/\*SDL\_RenderClear( gRenderer ); //Clear screen

SDL\_RenderCopy( gRenderer, gTexture, NULL, NULL ); //Render texture to screen

SDL\_RenderPresent( gRenderer );//Update screen\*/

SDL\_UpdateWindowSurface(window);

}

close();

}

bool initialize(){

initialize SDL{

}

create window(){

}

create renderer(){

}

initialize img\_loading{

}

return true;

}

bool load\_media(){}

void close(){

SDL\_DestoryTexture();

SDL\_DestroyRenderer();

SDL\_DestroyWindow();

free();

IMG\_QUIT();

SDL\_QUIT();

}

# Sound:

## Milestones:

### Milestone 1 – Play Background Sound

Due by March 20th. Background sound will begin as soon as the game begins. There will be different background sounds for the different stages of the game (menu, lobby, floors)

### Milestone 2 – Play sounds specific to an item/player

Due by April 3rd. Sounds will be played for the players/items collected. There will be more custom sounds such as a player compromising the system or a player being tagged.

## State Diagram:



## Pseudocode:

Load sound{  
}

Play Sound{  
}

# Coding Style/Choices:

The project will use a component-based model in C++. The motivation for this choice was to avoid the problems of adapting a large inheritance hierarchy to add features.

The library chosen for the game production is SDL2. Reasons are:

* SDL2 was written in C. This allows many of the conventions of C to be used to create the project.
* Portability. Due to the library being created in C, the game is quite portable to other systems with very little change in code. This portability allows the game to be played among different systems such as Windows, OSX and Linux.
* Wrapper functions. SDL2 has an abundance of wrapper functions to speed up the development of the project.
* Performance. SDL2 provides many performance boosts over other libraries such as C# because of the low level language use.

# Github Location:

The project used Git Hub for subversioning and the location of the github is:

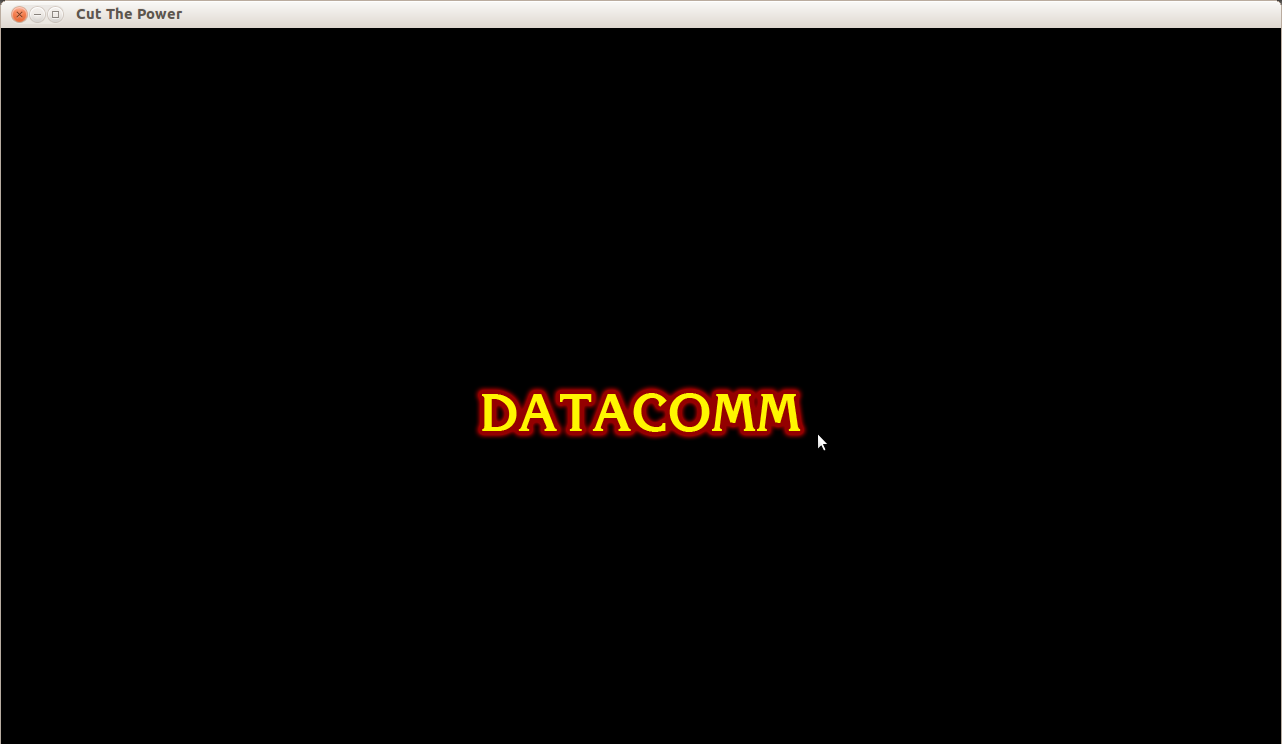
* https://github.com/DComm2013Network/CutThePower.git

# Screenshots:

The following are screen shots of the game. These images are up to date.

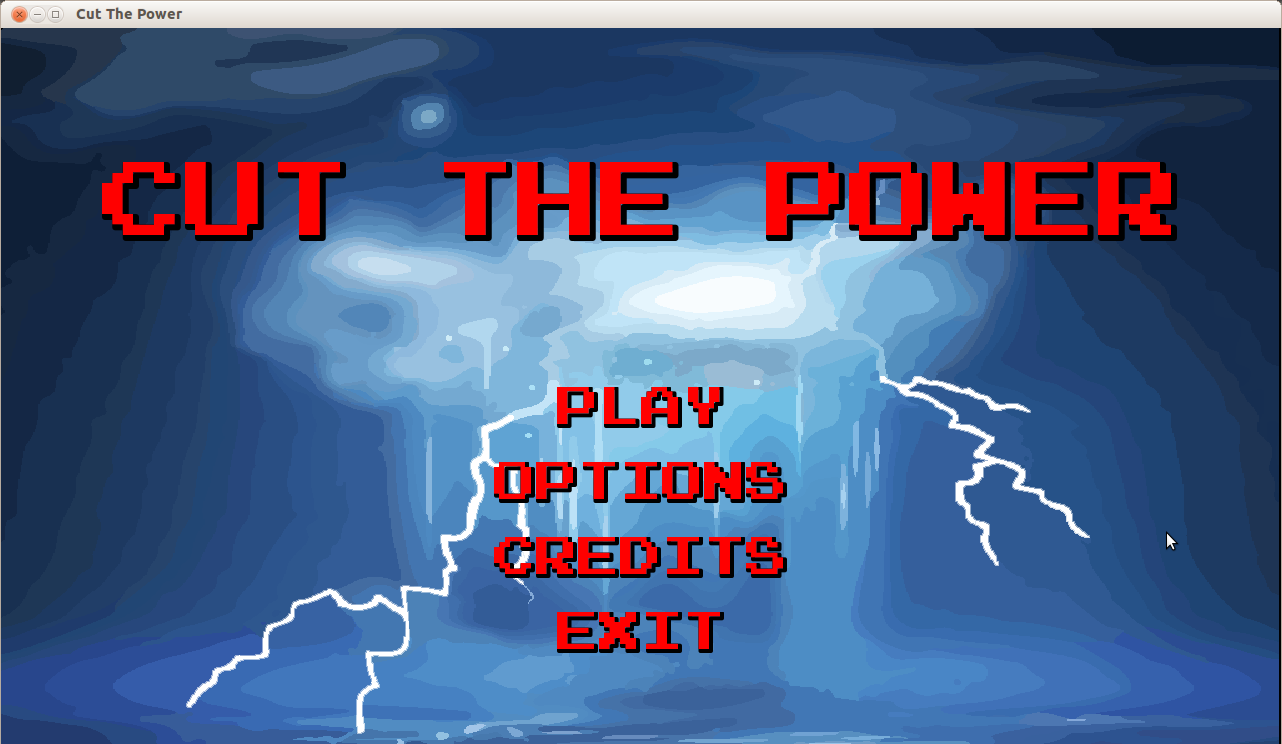
## Logo screen:

This is what happens when the game begins



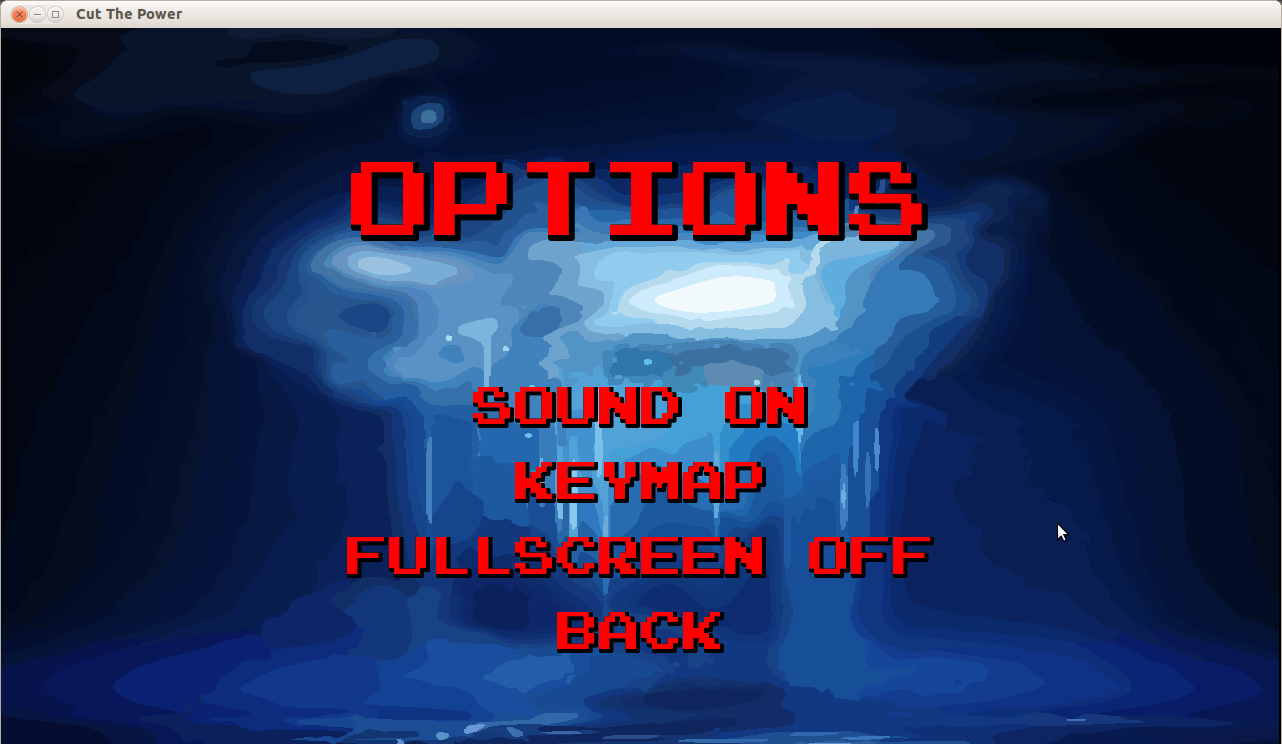
## Main Menu:

This is the main screen for the game. Users can play or customize options.



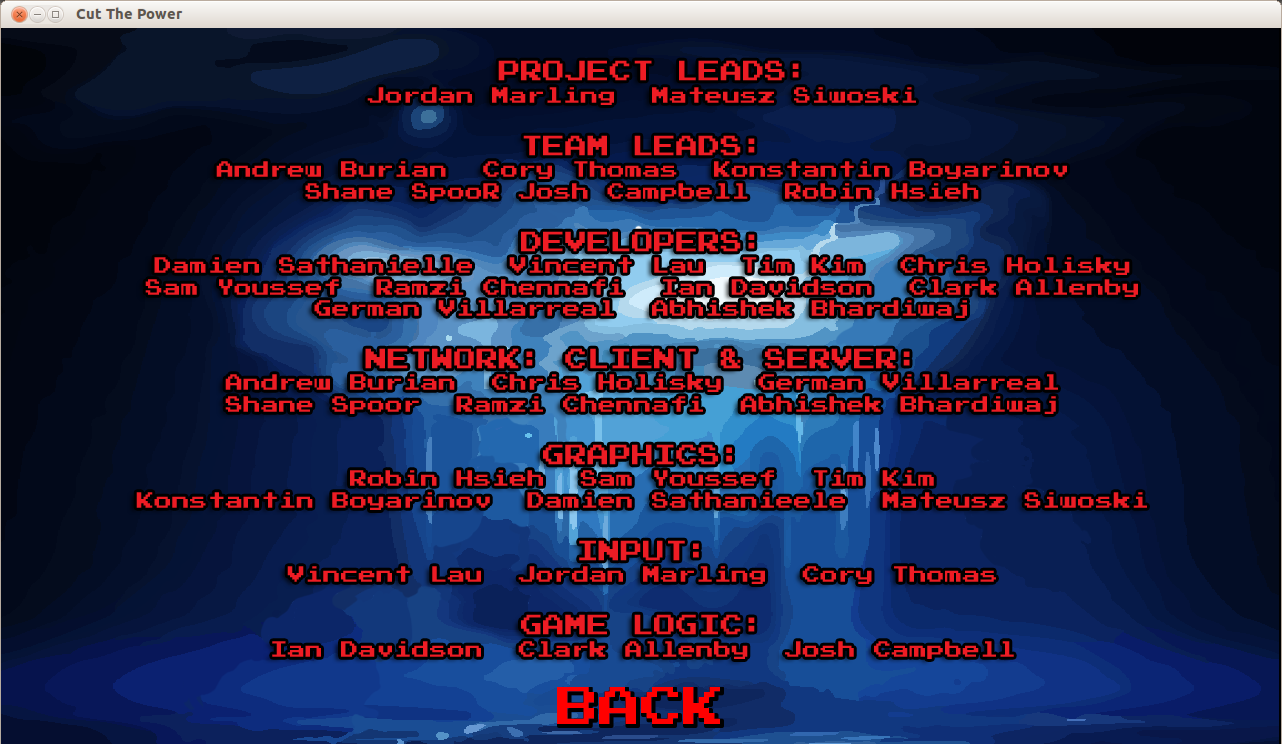
## Options:

Options screen where the users can customize the keyboard, turn sound on/off and enter/exit fullscreen.



## Credits:

Credit screen. The thunder animation plays on all the screens.



## Selection:

The various characters available. There is a 1 in 4 chance of selecting a special character. Sound plays when that occurs.



## Server IP Address/Username:

This is where the username is entered and the IP address of the server. If the username is less than 3 characters or invalid, the server assigns a username. IP address can be either LAN or WAN.



## Driver Animation:

Character animation that plays after a character is selected. Custom to each character.



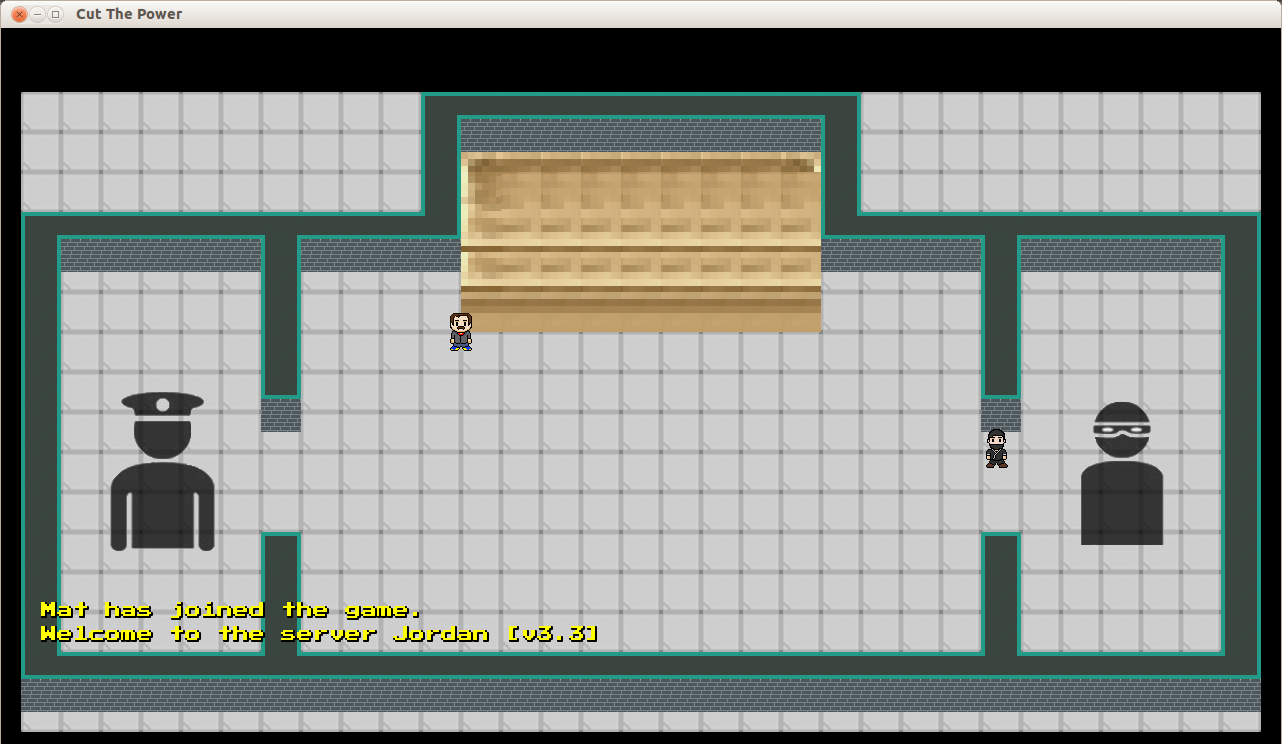
## City Animation:

After the car animation.



## Lobby:

This is the beginning of the game. This is where all users will be able to enter and play. They will be able to choose a side they would like to play as. Chat works during this stage. When both sides are chosen, a countdown begins to the start of the game.



## Floors:

These are the various floors of the game in which power ups and servers are selectable throughout the map. Chat works among these stages.

### Parking Lot:



### Floor 1:



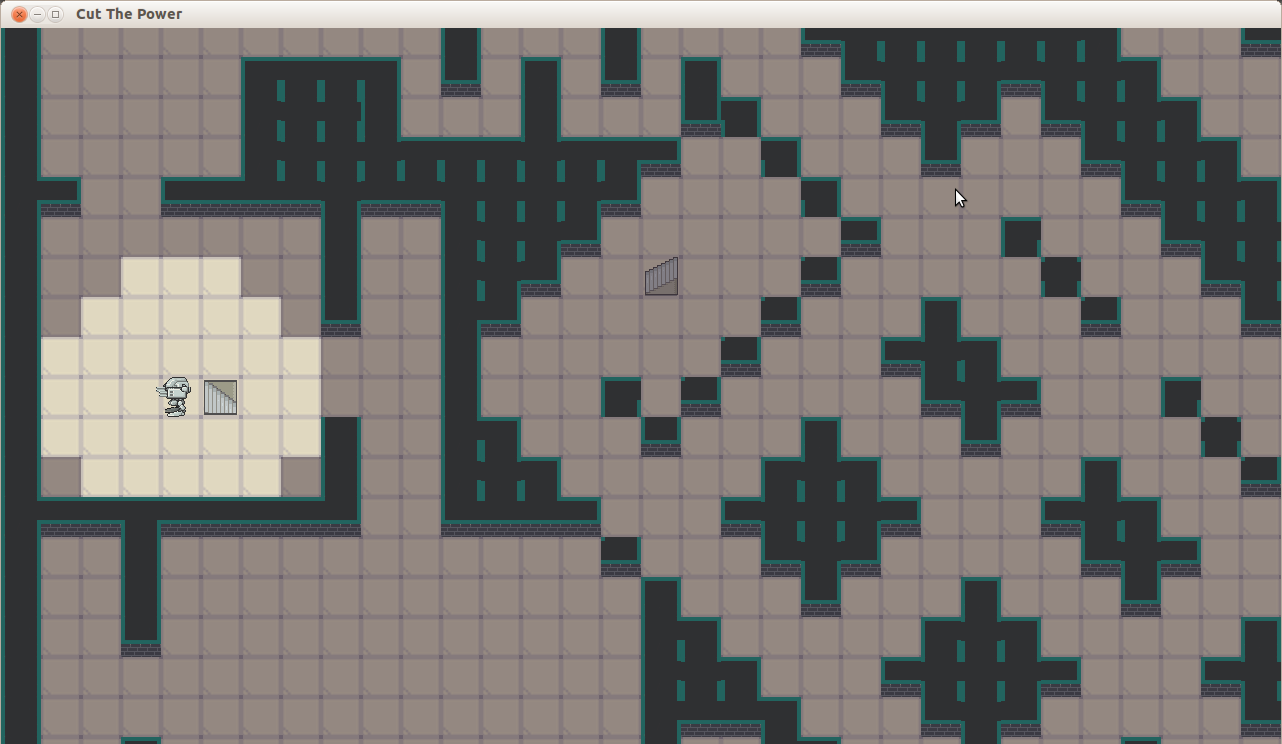
### Floor 2:



### Floor 3:



### Floor 4:



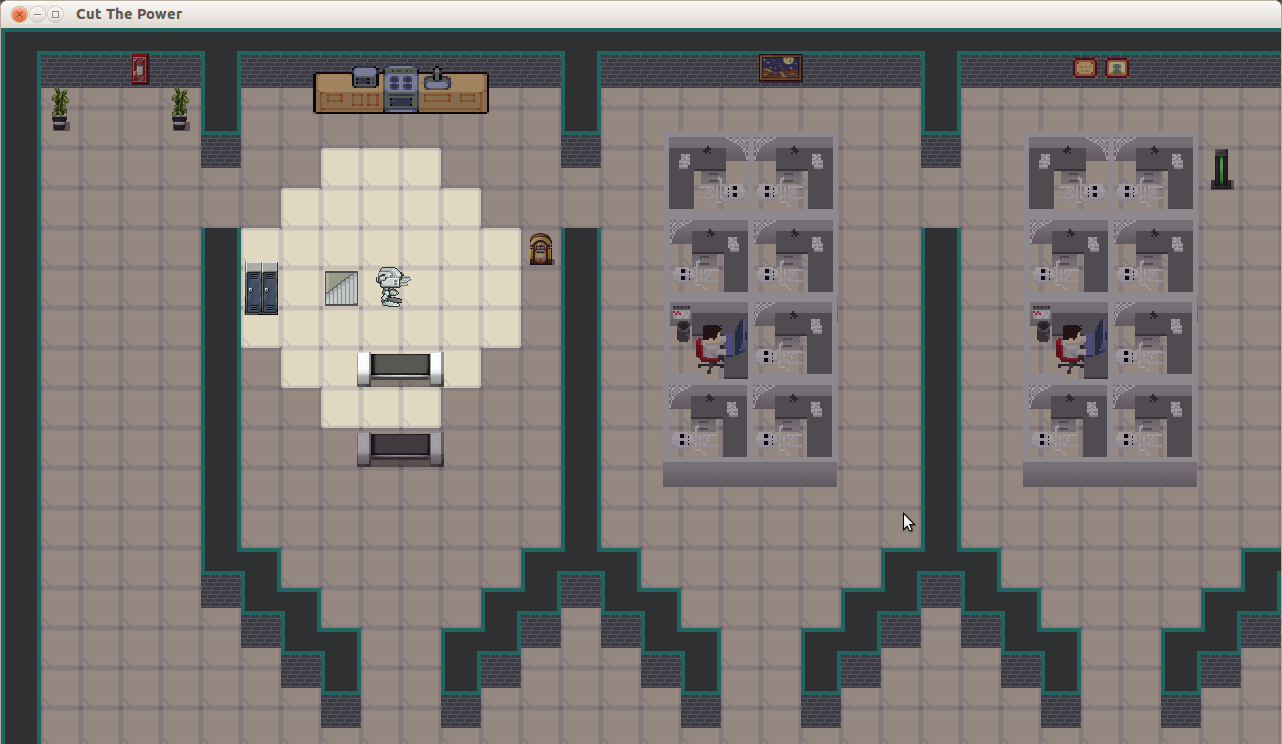
### Floor 5:



### Floor 6:



### Floor 7:



### Floor 8:

