LightBikes

Documentation

Team 3 – Derezzed Games

Felice Aprile

Justin W. Flory

Malcolm Jones

Timothy Endersby

Date: 13-05-2016

unaccepted Revision: 0.2

Table of Contents

Executive overview **3**

Audience **3**

Assumptions made for this project **3-4**

Gantt chart **5**

Gantt task descriptions: *5*

Class and method overview **6**

LightBikes UML **13**

Client GUI **13**

Protocols **14**

Data used **15**

Punch List used **16**

To do: *16*

Done: *16*

Unresolved Issues **17**

Revision History

Name	Date	Reason for Change	Version
Justin W. Flory	13-05-16	Initial draft	0.1
Justin W. Flory	17-05-16	Updating based on feedback in class	0.2

Executive overview

LightBikes is a multiplayer game between two people. In the game, both players are controlling a "bike" across a grid-based area. As the player moves across the board, they leave a wall of "light" behind them. If a player runs into the light wall (either their own or the opponent's), they will lose the game. This game is inspired by the same game from the TRON movie.

Audience

This design document is intended for project managers, current employees working on the project, and future employees who may work on this project. The document covers the design decisions made for the game, provides an overview of the tasks for the project, and other milestones related to the project.

The document will give an overview of the game and provide greater understanding for someone to see how the project works and how it is designed.

Application intentions

LightBikes is targeted towards children and young adults that play video games. Our game intends to provide entertainment and enjoyment to those who play it, as well as to help promote the game development by Derezzed Games across all of our products.

There is also potential for the game to have a revenue stream in the form of advertising and micro-transactions. The game could support embedded advertisements while users are playing or the platform the game is deployed to could use advertisements. Alternatively, through the form of “upgrades” to either the bike or arena, the game could support micro-transactions to add new dynamics into the gameplay.

Assumptions made for this project

The controls for the game will use the arrow keys to control the movement of the bike. The corresponding direction will change the direction of the bike. Pressing the spacebar will start the game or restart the game if you just ended one.

When a bike is moving, it leaves a pathway behind it. As the bike moves across the board, the pathway grows longer and longer. When any bike touches the “wall” left

behind by other bikes (or its own), the player will lose a life. When they lose a life, all of the walls left by them will disappear.

The majority of the game logic will be on the client-side of the program. The role of the server in the game is to only act as a relay between multiple clients and pass as little data as possible between all the clients.

The client will perform actions and simulate the actions of other players locally in the client. The client will send the movement history of the current player in the form of directions across the network. The client will keep drawing the wall until it receives notice of a direction change of the opponent from the server.

Gantt chart

For our project, we used a Gantt chart for our project management and organizing tasks. We used the free and open source tool GanttProject for our work. The tasks from our chart are broken down below.

Gantt task descriptions

Begin Date	Complete?	Tasks	End Date
2016-03-28	X	Brainstorm on project ideas	2016-03-29
2016-03-29	X	Select project idea to use	2016-03-30
2016-03-30	X	Brainstorm logistics and how game will work	2016-04-01
2016-04-01	X	Complete initial draft of design document	2016-04-04
2016-04-04	X	Project planning, blueprinting, white boarding	2016-04-08
2016-04-13	X	Create initial framework for GUI	2016-04-14
2016-04-18	X	Create keypress events for bike	2016-04-18
2016-04-18	X	Create initial framework for GameServer	2016-04-25
2016-04-18	X	Create grid for the lightbikes	2016-04-25
2016-04-18	X	Ask for user settings in GUI for chat	2016-04-18
2016-04-18	X	Optimize location of chat in GUI	2016-04-18
2016-04-18	X	Functioning menu bars	2016-04-18
2016-04-20	X	Remove hard-coded functionality in chat server	2016-04-22
2016-04-21	X	Implement bike movement	2016-04-25
2016-04-25	X	Integrating client with server network	2016-05-02
2016-04-25	X	Handling collisions	2016-04-29
2016-04-25	X	Establish communications between client/server	2016-04-25
2016-05-02	X	Polish existing GUI	2016-05-13
2016-05-09	X	Begin updating Design Document	2016-05-10
2016-05-09		Packaging client and server JARs	2016-05-11
2016-05-09	X	Make KeyListeners work	2016-05-11
2016-05-09	X	Integrate network with game logic	2016-05-11
2016-05-10	X	Write user documentation	2016-05-12
2016-05-11	X	Create project presentation	2016-05-11
2016-05-12	X	Apply JavaDocs comments to project (where missing)	2016-05-12
2016-05-12	X	Take screenshots of working game demo	2016-05-12
2016-05-12	X	Create a server GUI	2016-05-13
2016-05-13	X	Check in with professor on project status	2016-05-13
2016-05-13	X	Expand on existing JavaDocs	2016-05-16
2016-05-13	X	Update design document with feedback from professor	2016-05-16

Begin Date	Complete?	Tasks	End Date
2016-05-17	X	Add info about gameplay instructions	2016-05-17
2016-05-18	X	Create a live demo video	2016-05-18
2016-05-18	X	Final team check-in	2016-05-18
2016-05-19		Present project at trade show	2016-05-19

Class and method overview

Overview of the classes and functionality.

Chat Server- Server to manage Chat functionality

- Constructor
    - Sets up Network sockets, and waits for clients
- Inner Class ConnectedClient
    - Constructor(Socket sock)
        - Creates Client thread
    - Send(String msg)
        - Sends message
    - sendToAll(String msg)
        - Send message to all clients
    - run()
        - Manages getting and sending messages

Game Server – Manages network functionality of game itself

- Main Method
    - Creates Game Server Object
- Constructor
    - Creates a gui for server, waits for client connection
- pushToAll(String commandString)
    - sends command to all clients
- pushToPlayer(int playerId, String commandString)

    - sends command to specific player
- pushToOthers(int playerID, String commandString)
    - sends commands to players that aren't playerID
- makeCommandString(String command, String value)
    - Creates commandString for other methods
- startGame()
    - sends command to all clients to start a game

Player – Server side handler of Client-sent commands

- Constructor(Socket s, int playerID, GameServer gameServer, JTextArea playerOutput)
    - Passes values to player object
- run()
    - Handles sending player id, and starting game
- push(String line)
    - send message to client
- push(String command, String value)
    - sends command to client
- pushToOthers(String command, String value)
    - sends command to other clients
- pushToAll(String command, String value)
    - sends command to ALL clients
- makeCommandString(String command, String value)
    - makes command string to send
- processCommand(String cmdString)
    - handles input command depending on command
- setUsername(String username)
    - setter for Username

- getPlayerID()
    - returns playerID
- getUsername()
    - returns Username
- sendPlayerID()
    - sends player ID out
- inner Listener Class
    - Constructor(Socket s, Player p)
        - Creates Listener for player in a socket
    - Run()
        - Listens for commands to parse
    - parseCommands(String line)
        - parses command

ChatClient- Client for chat window

- Constructor(String username, JTextArea chat, JTextField newMsg)
    - Creates gui, and sets up an actionListener
- Connect()
    - Creates sockets and reader/writer
- Send(String msg)
    - Sends message over printwriter
- closeSocket()
    - Closes everything
- Inner Class RecieveMessages
    - Run()
        - Appends read line into the chat window

Network Connector – Connects server to clients with commands

- Constructor(String hostname, String username, Grid grid)
    - Passes variables and connects
- Connect()
    - Connects lightbikes client to server with username and hostname
- sendUsername(String username)
    - updates username to Server
- sendLocation(int x, int y)
    - sends Location to server via x and y
- notifyDeath()
    - tells server that client died
- send(String commandString)
    - Sends commandString to server
- sendCommand(String command, String value)/(String command, int value)
    - Creates commandString, then sends it
- makeCommandString(String command, String value)
    - Makes CommandString from command + value
- processCommand(String cmdString)
    - Handles commandString depending on value
- startGame(String value)
    - starts game for client
- setPlayers(String value)
    - Process the list of usernames sent by the LightBikes server and store them for later use.
- updateLocation(String value)
    - updates other bikes position in user's client
- setUserID(String value)
    - setter for UserID

- getUserID()
    - getter for UserID
- Inner class Listener – handles server sending data
    - Constructor(Socket s)
        - Opens socket and listens
    - Run()
        - Waits for commands and parses
    - parseCommands(String line)
        - processes commands

Bike – Bike object class

- Constructor()
    - Passes values
- turnWest/East/South/North()
    - changes direction, disallows 180 degree turn
- stop()
    - declares player now
- checkLocation(int x, int y)
    - checks to see if bike collides with border or wall
- updateLocation()
    - moves players
- setLocation(int x, int y)
    - sets player location and updates
- getGameState/xPos/yPos
    - getter methods
- startGame()
    - creates a Movement Thread
- Inner Class Movement – handles movement

    - Run()
        - Moves piece based off direction, and handles death

Grid – Game board

- Constructor()
    - Creates a 2d array and sets it to 0
- startGame(int controlled)
    - creates bikes and sets whichever one you can drive
- connect(String hostname, String username)
    - creates a NetworkConnector for the Client
- turnNorth/East/South/West()
    - Passes movement commands to bike
- Stop()
    - Passes stop command to bike
- Won/lost()
    - Shows a win or loss message
- getServerBike()
    - returns bike being controlled by other client
- getConnector()
    - returns NetworkConnector
- paintComponent()
    - Repaints grid to draw player walls at every repaint

LightBike – Main game class

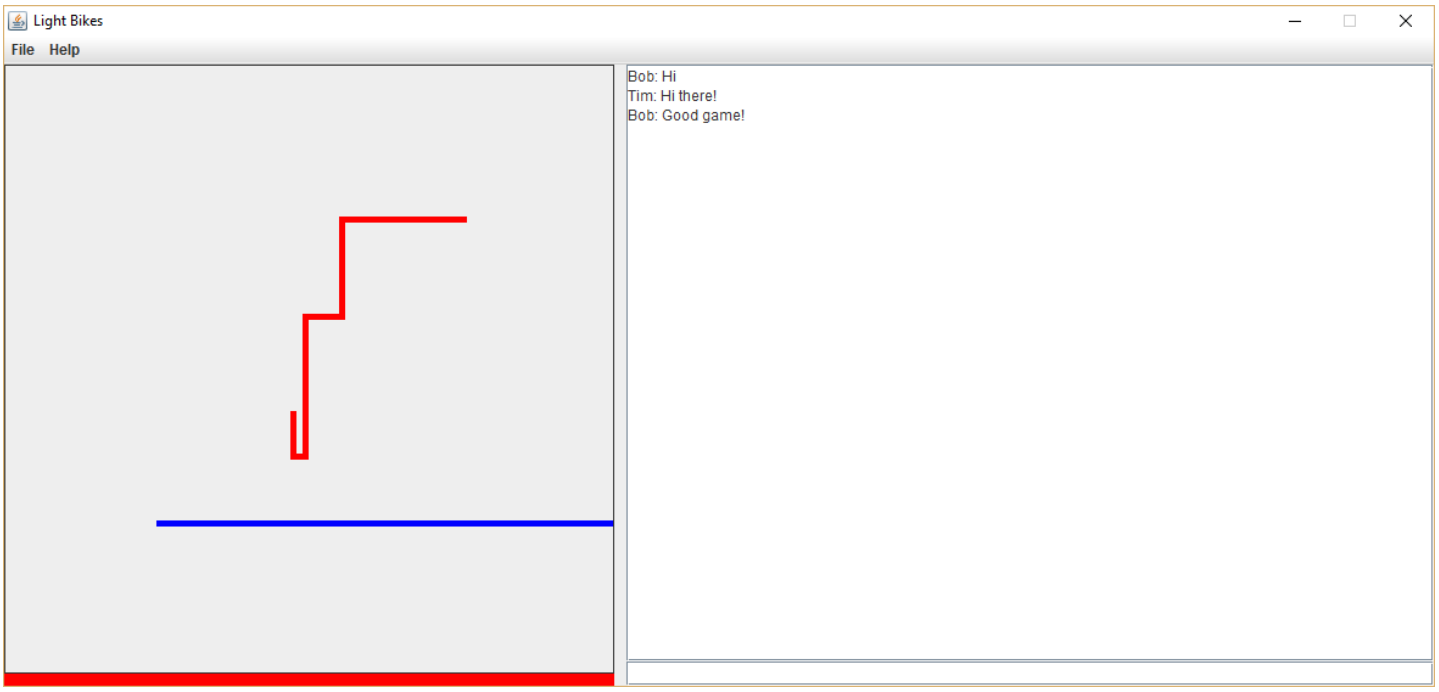
- Constructor
    - Creates gui and add listener, as well as asking for username and hostname
- keyPressed
    - handles keypress movement

- mouseEntered()
    - allows player control after using chat

LightBikes UML

UML files are attached separately from this document (as PNG images).

Client GUI



Networking connections & Protocols

- **Client connection to server**: Client specifies IP address (or hostname) to connect to
- **Port number(s)**: 6667 used because of its association with the IRC chat protocol (RFC 2812) which was used for the chat server in mind
- **Protocol interface code for both client and server**:
  - No interfaces or abstract methods used
  - Constant used for defining game version displayed to user

Communication class

Chat interaction between Clients and Server.

Server starts first and waits for two client connections. Multiple clients connect to the server. First two are players, others only have chat privileges. When message sent to server from a client, the message is sent back to all clients. Message is displayed on the GUI.

Client	Communication	Server
		Startup
		Waits for client to connect
Client connection	Connection, no data →	Accept connection
Client sends chat msg	Message →	Server reads info
Client receives chat message	← Message to all	Sends message to all client(s)
Client shows it on HUI		

Data used

No external data is used in the project.

Data files

No external files or libraries were used for the completion of our project

Punch List used

To do:

- [all items complete]

Done:

- Work out an initial design document and fill it as we go
- Begin creating the basic framework for the project to cookie-cutter template the game
- Get an early GUI running
- Begin implementing game logic
- Start communication between server and clients
  - Including chat
- Clean up GUI with game logic and networking
- Begin tying the details together (see more info in Gantt)
- Better document code as we're going, make it easier to maintain
- Update Gantt and punch list for project

Unresolved Issues

- Spectators should also be able to view the existing game
- Support for more than two players?
- Some computers seem to lag more than others, more research and investigation could go into figuring out if there are unaccounted variables on different platforms and ways we could optimize