System Design Document

Group: AssadFC

CRC Cards

Player	
• Knows its position (x,y coordinate) • Knows its horizontal velocity • knows its vertical velocity • Knows the number of health points • Carries Gun • Shoots Gun • Jumps • Moves	• Gun

State Machine	
• Contains lists of State objects • Knows current State • Changes to a different State • Exits current State • Updates current State • Draws current State	• State

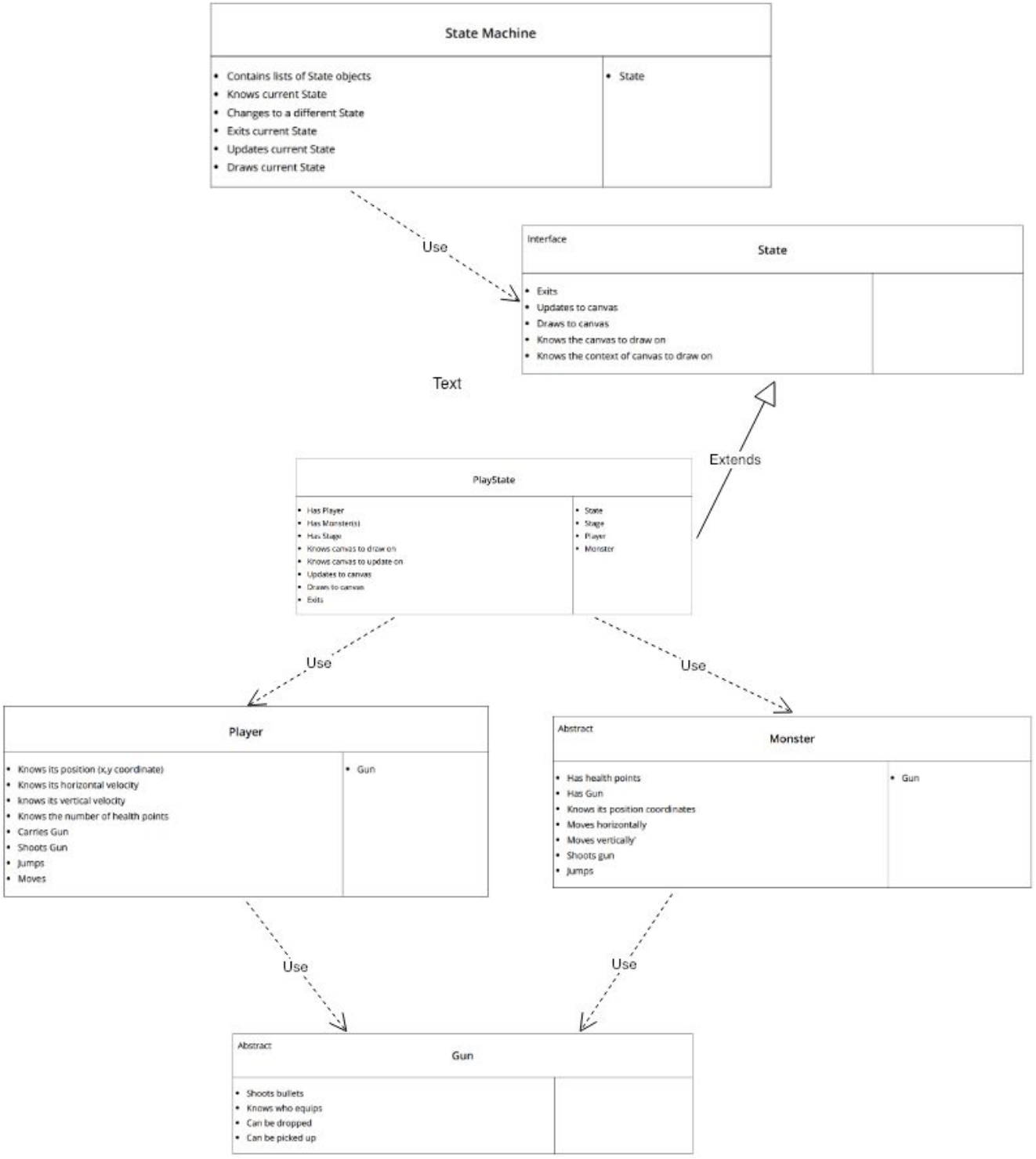
Interface State	
• Exits • Updates to canvas • Draws to canvas • Knows the canvas to draw on • Knows the context of canvas to draw on	

PlayState	
• Has Player • Has Monster(s) • Has Stage • Knows canvas to draw on • Knows canvas to update on • Updates to canvas • Draws to canvas • Exits	• State • Stage • Player • Monster

Abstract Monster	
• Has health points • Has Gun • Knows its position coordinates • Moves horizontally • Moves vertically' • Shoots gun • Jumps	• Gun

Abstract Gun	
• Shoots bullets • Knows who equips • Can be dropped • Can be picked up	

CRC Model


State Machine
Contains lists of State objects
Knows current State
Changes to a different State
Exits current State
Updates current State
Draws current State
State
Use
Interface
State
Exits
Updates to canvas
Draws to canvas
Knows the canvas to draw on
Knows the context of canvas to draw on
Text
Extends
PlayState
Has Player
Has Monster(s)
Has Stage
Knows canvas to draw on
Knows canvas to update on
Updates to canvas
Draws to canvas
Exits
State
Stage
Player
Monster
Use
Use
Player
Knows its position (x,y coordinate)
Knows its horizontal velocity
knows its vertical velocity
Knows the number of health points
Carries Gun
Shoots Gun
Jumps
Moves
Gun
Abstract
Monster
Has health points
Has Gun
Knows its position coordinates
Moves horizontally
Moves vertically
Shoots gun
Jumps
Gun
Use
Use
Abstract
Gun
Shoots bullets
Knows who equips
Can be dropped
Can be picked up

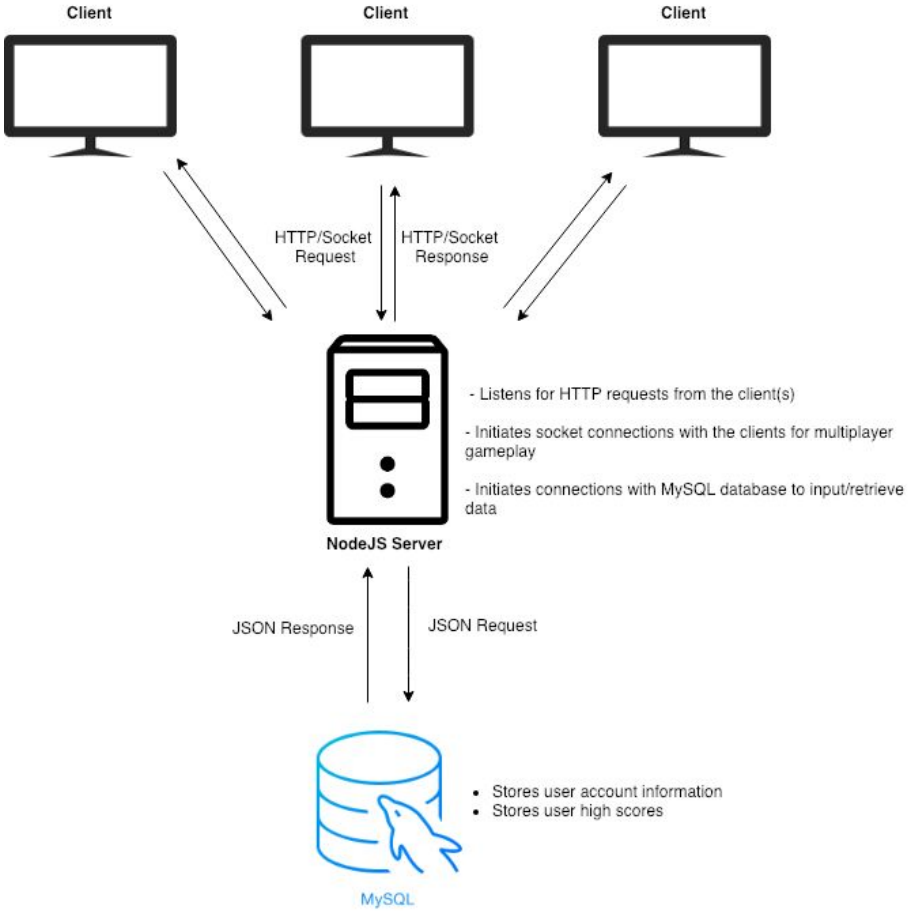
System Interaction and Environment

The main components and technologies the our “Untitled” game will have are:

- Server: NodeJS, Express and Socket.IO (Javascript libraries)
- Database: MySQL (tentative)
- Front-end: HTML, CSS and Javascript. Libraries such as Express and Socket.IO will also be used in certain states of the game.

Since this is a web browser game, the only requirement is that the user owns a PC with an operating system that supports Chrome, Firefox, Edge and Safari. These browsers support essential Javascript functions that allow the user to play the game, such as event handlers for key and mouse movement inputs. Moreover, we have not considered implementing the game for mobile devices, since mouse movements are a core mechanic in the game.

System Architecture



System Decomposition

The main components of our “Untitled” web browser game are the client(s), the NodeJS server and the MySQL database. The architecture is describes using the model-view-controller (MVC) design.

- **The client (view)**

The client(s) will send HTTP request to the NodeJS server, which in turn will respond with the HTML, CSS, JavaScript and any other dependencies the game requires to run. Client-side code is designed as a finite-state machine (FSM), where each state is a different visual component that a user interacts with. For example, the first state (“MenuState”) would be the main menu the user interacts with when first logged into the website. These are defined as classes, such as State Machine class, which includes the state classes (i.e. MenuState, PlayState, and more). These states would normally inherit from a BaseState interface, but implementation of an interface in javaScript is not possible or at least complicated. Moreover, there are classes included on certain states that directly correspond to the gameplay, such as Player, Monster(s), Missile and GameItem classes.

- **The server**

The main purpose of the server, besides the hosting of the main files of the game, is to manage socket connections with multiple clients. For example, if a user selects multiplayer mode on MenuState, it would attempt to connect to the server’s open listening socket. During gameplay, any event triggered by the client would be sent to the server via the socket in JSON (or possible another) format. The server would then emit the data to all the clients if and only if those clients are in the same game as the other client. Another important duty the server must do is handle inputs for user registration (using proper password sanitization). The server would process those inputs and send them, using JSON format, to the MySQL database for storage. Whenever the server requires data, such as display of high scores for multiplayer users, it shall query the database for the data accordingly. We anticipate that there might network connectivity issues such as lag between the clients and servers, as well as total connection loss. For the former, missile trajectory predictions would be possible, albeit quite difficult. For the latter, we can ask the user to refresh the page and allow him to maintain his previous session of play.

- **The database**

The MySQL database will store any game-related data for each user that registers. For example, each user will have a unique ID and have the following attributes: User, Password (hash), Date of Creation, Score (game points).