<u>CSCI 4448 - Project Part 2</u>

Team: Brady Auen, Nicholas Nocella, Peter Huynh
Title: Type-Speed Game

Project Summary:

Type-Speed is a Java-based game designed to test the player's typing skills. Users will type words that fly by the window and receive a score for each word that they type correctly. Players will also have the option to choose a difficulty speed for the the flying words and view the overall high scores of the game in a separate menu.

Project Requirements:

Business Requirements

ID	Requirement	Topic Area	User	Priority
BR-01	Opening the game starts at a selection menu where you can access the game, scores, and difficulty. It will be minimalistic, with splash art that can draw in an audience.	Game	Player	High
BR-02	Anyone can have access to the game, scores, and difficulty when inside the game. The game will be available to the public.	Game	Player	Medium

User Requirements

ID	Description	Agile Sizing	Priority
UR-01	As the player, I want to be able to open the game and see the starting menu options	Medium	High
UR-02	As the player, I want to be able to press the difficulty button and change the difficulty.	Small	Low
UR-03	As the teacher, I want to be able to press the high score button and view the top 10 scores	Small	Low
UR-04	As the teacher, I want to see the words per minute of each high score when I view the high scores	Small	Low

UR-05	As the player, I want to be able to press the play button and go into the actual game.	Large	High
UR-06	As the player, I want to see my input print onto the top of the screen when I press a key in the game.	Medium	Medium
UR-07	As the player, I want to submit the keys I pressed before as a word when I press the enter key.	Small	Medium
UR-08	As the player, I want to delete the keys I pressed before as a word when I press the backspace key.	Small	Medium

Functional Requirements

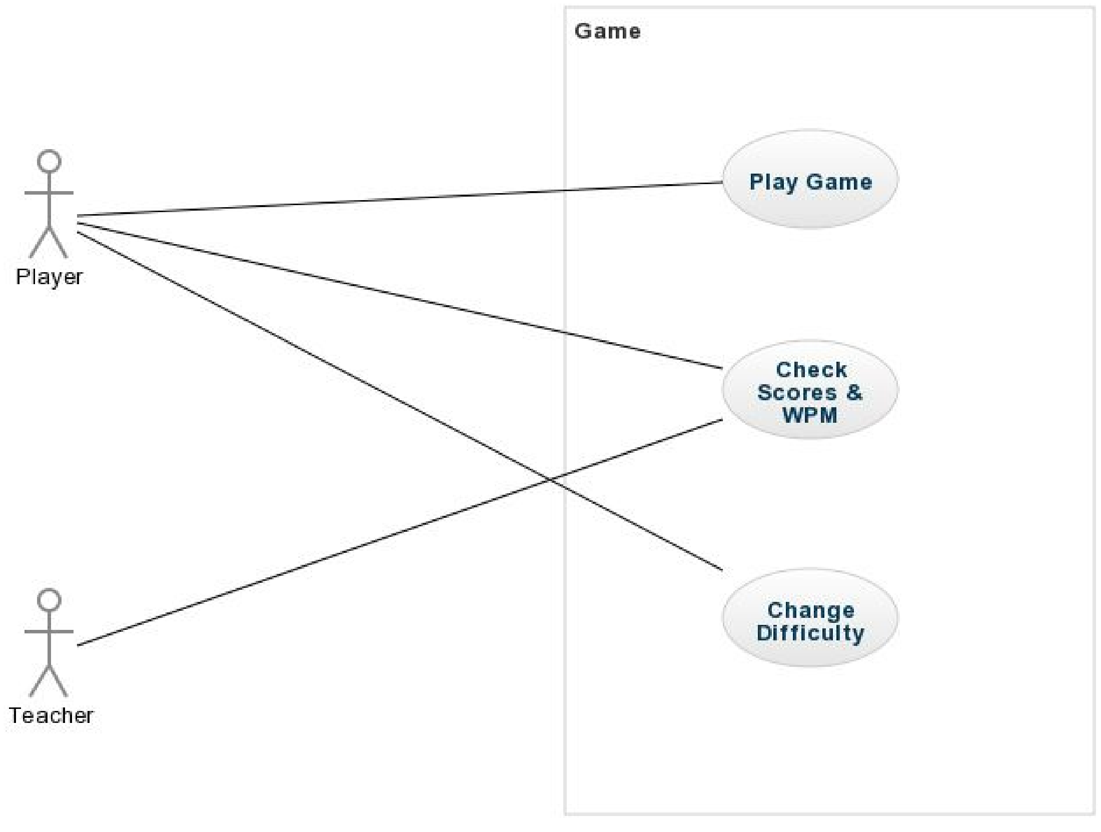
ID	Requirement	Priority
FR-01	When the difficulty is changed, the speed of words flying by in the game will change. Difficulties will be different states.	Low
FR-02	Each game state is separated and independent from one another.	Medium
FR-03	Words inside the game state “game” will fly horizontally and disappears when the player correctly input them.	High
FR-04	Pressing any button inside the menu state will change the state of the game.	Medium
FR-05	The game state “game” or “level” will use a word reader class to read a text file full of words separated into lines and use the words as the game words	High
FR-06	Scores will be recorded within a text file along with words per count, which will then be displayed in the scores state	Medium

Non-Functional Requirements

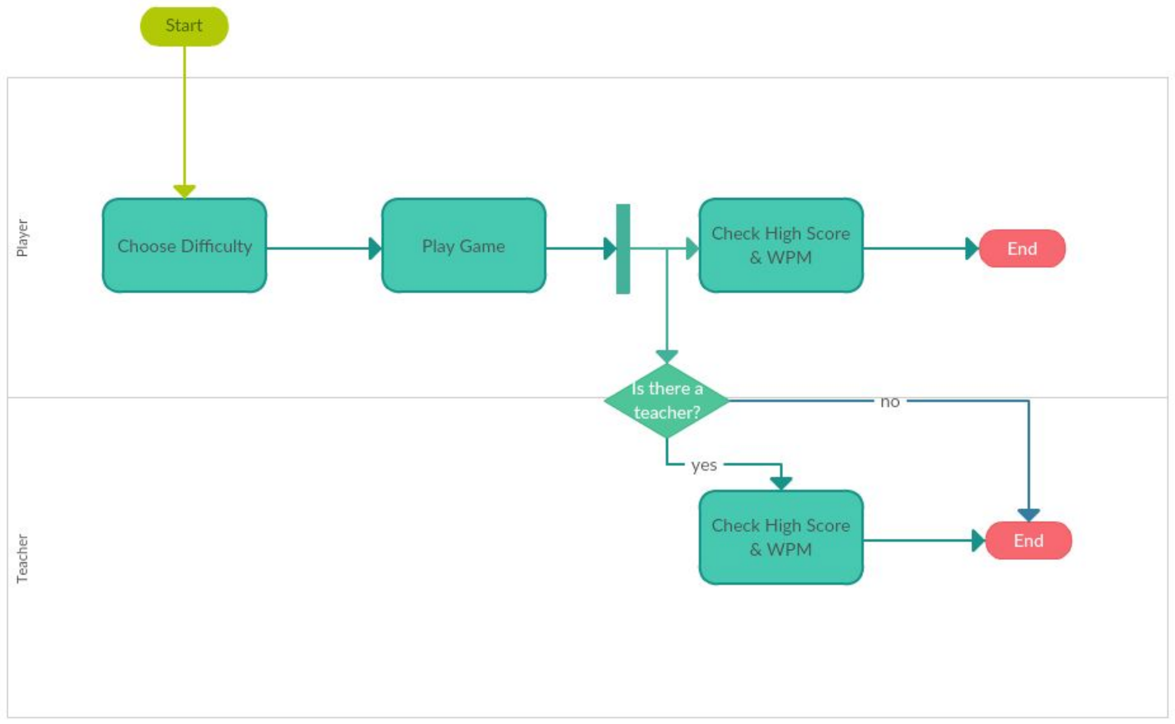
ID	Requirement	Priority
NFR-01	Everything is contained within one executable file as the finished product.	High
NFR-02	Data files will be inside text files, and art asset inside png files.	Medium
NFR-03	We will use the Lightweight Java Game Library (lwjgl) for the GUI.	Low

User and Tasks:

We envision our system possibly having two different users. One is a player, who'll want to play the game, check his/her high scores, and change the difficulty. Another user possible for this envisionment, but won't necessarily be required, is a teacher. The teacher will have access to look into the player's score and typing skills. Players use the game as a way of type practicing, and other people can possible view the results of such practicing. If the teacher or player desires so, he/she can evaluate the player's words per minute (WPM) through the high scores. Both would simply have to check the high scores board, which will include the player's WPM next to the total score.



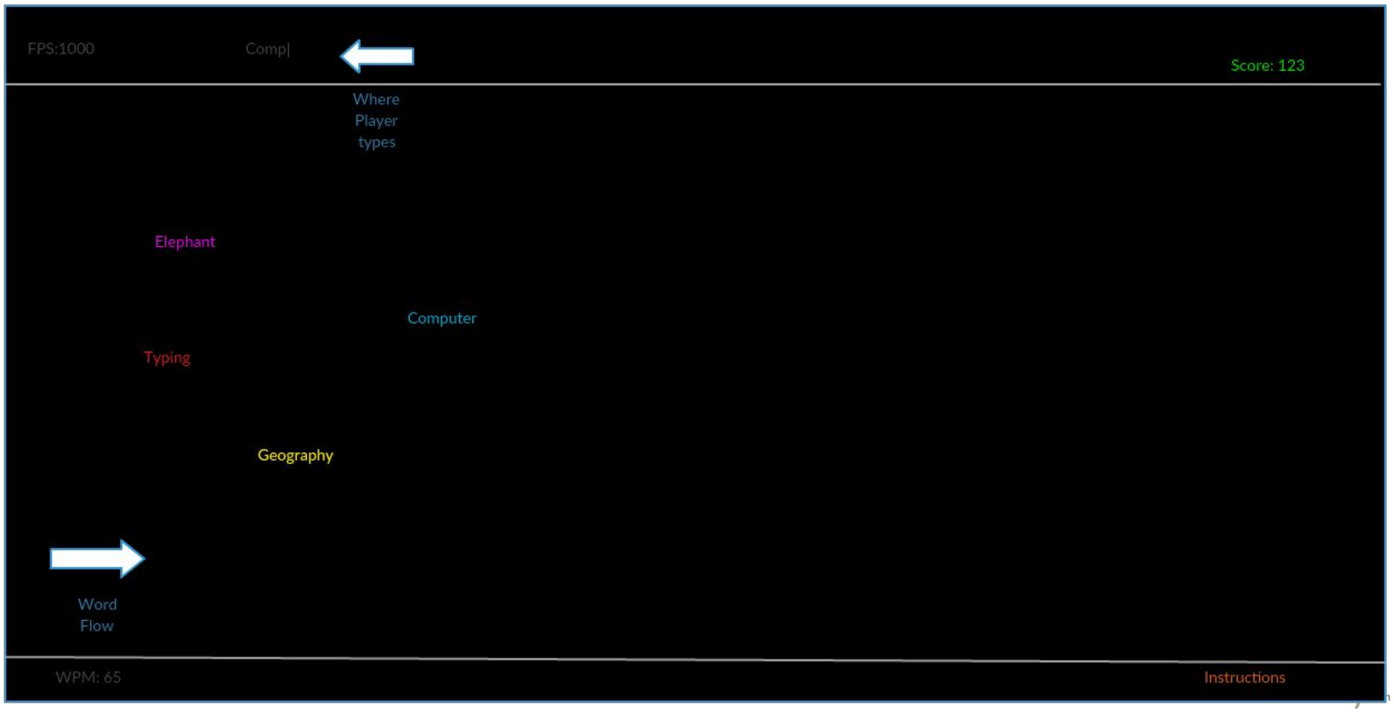
Activity Diagrams:



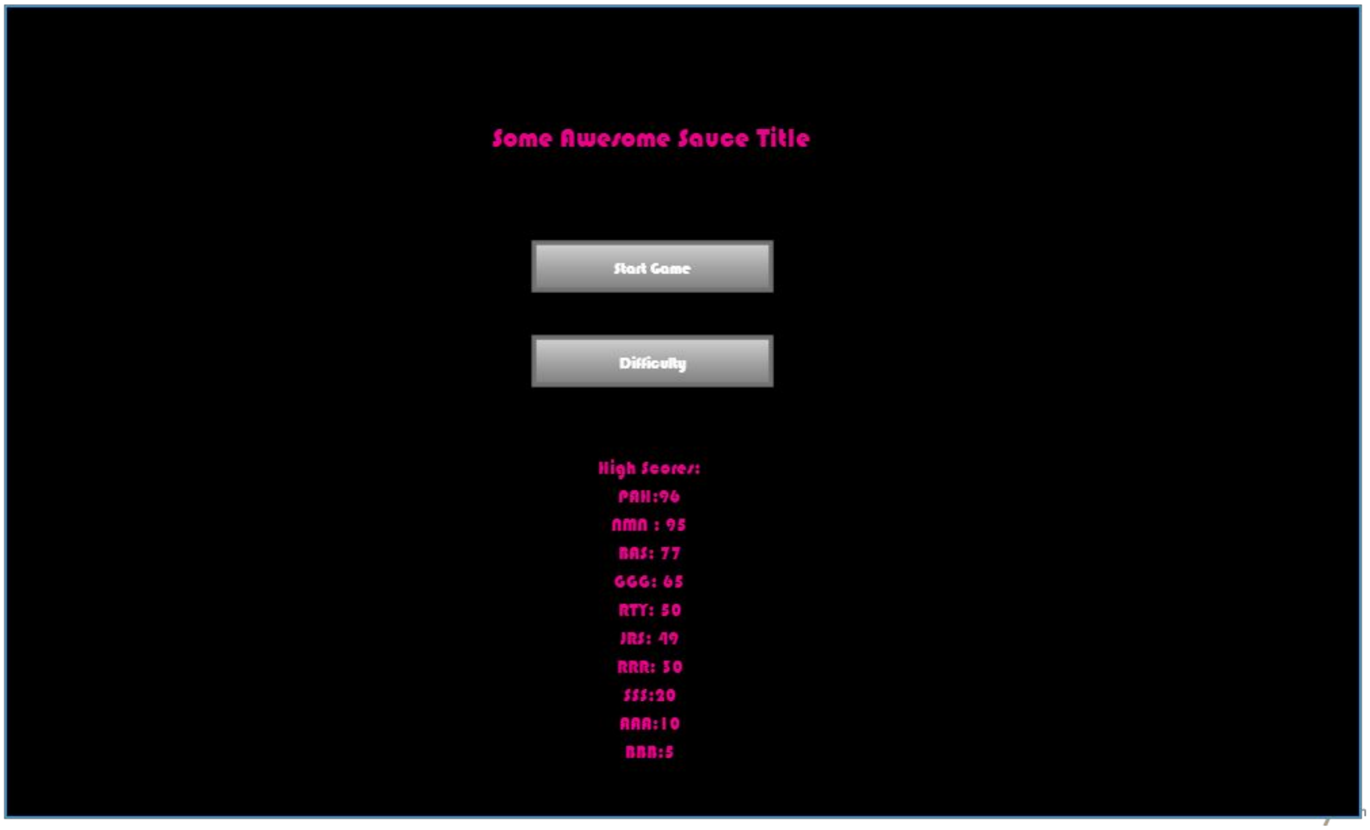
Data Storage:

For our game we will be using text files to store our data. One text file will store all the words used in the game as a dictionary. Words are simply listed in alphabetical order each on their own line. Another text file will be used to store high scores of the game. At runtime the WordReader class uses the dictionary of words. Images and art assets will be stored a png files, but will be rendered mostly in the MenuGui class. The game's gameplay will not render any art assets, but will use the Lightweight Java Game Library Slick2D to render polygons and text not stored as data. Rendering during the gameplay is all done during execution.

UI Mock-Ups: Game UI Mockup



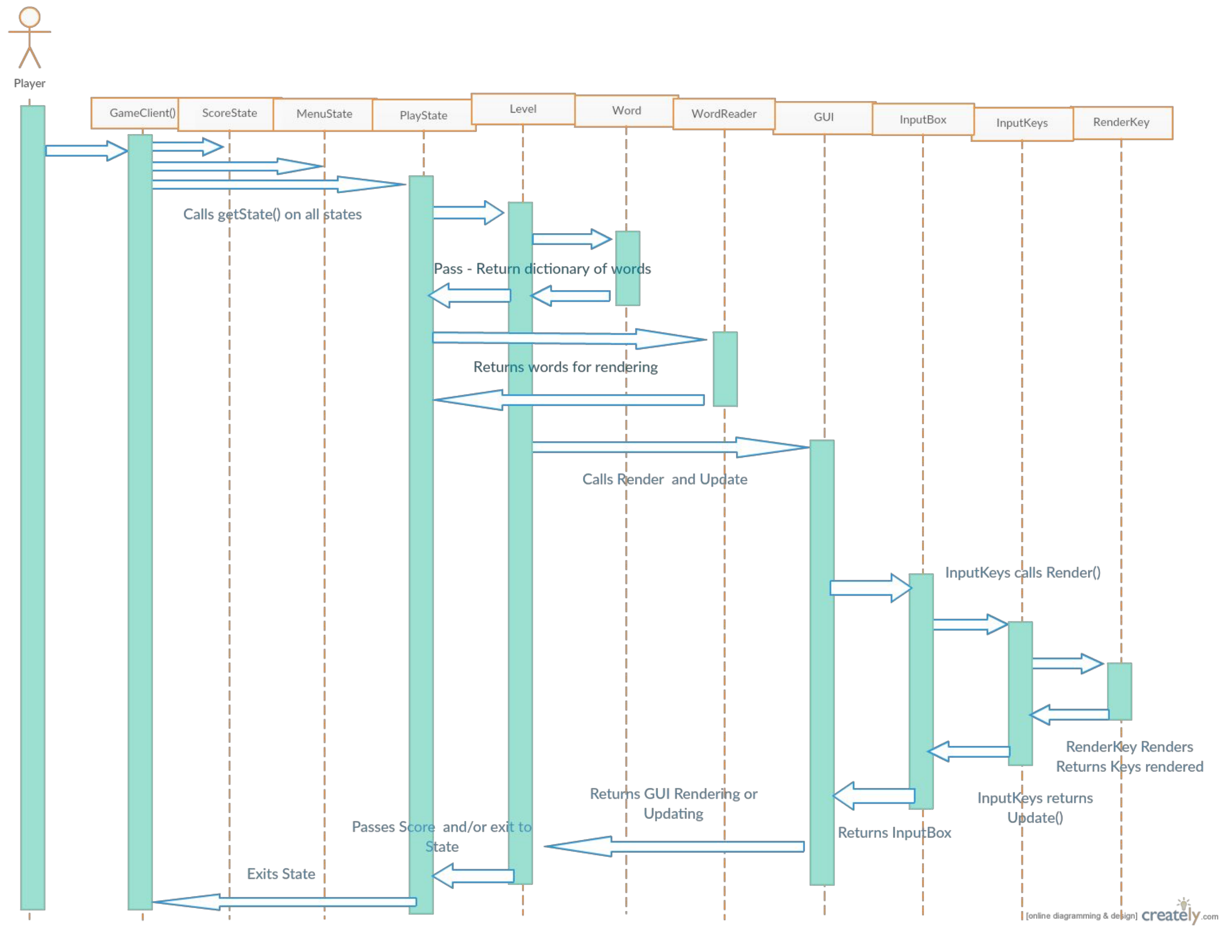
Menu Mockup



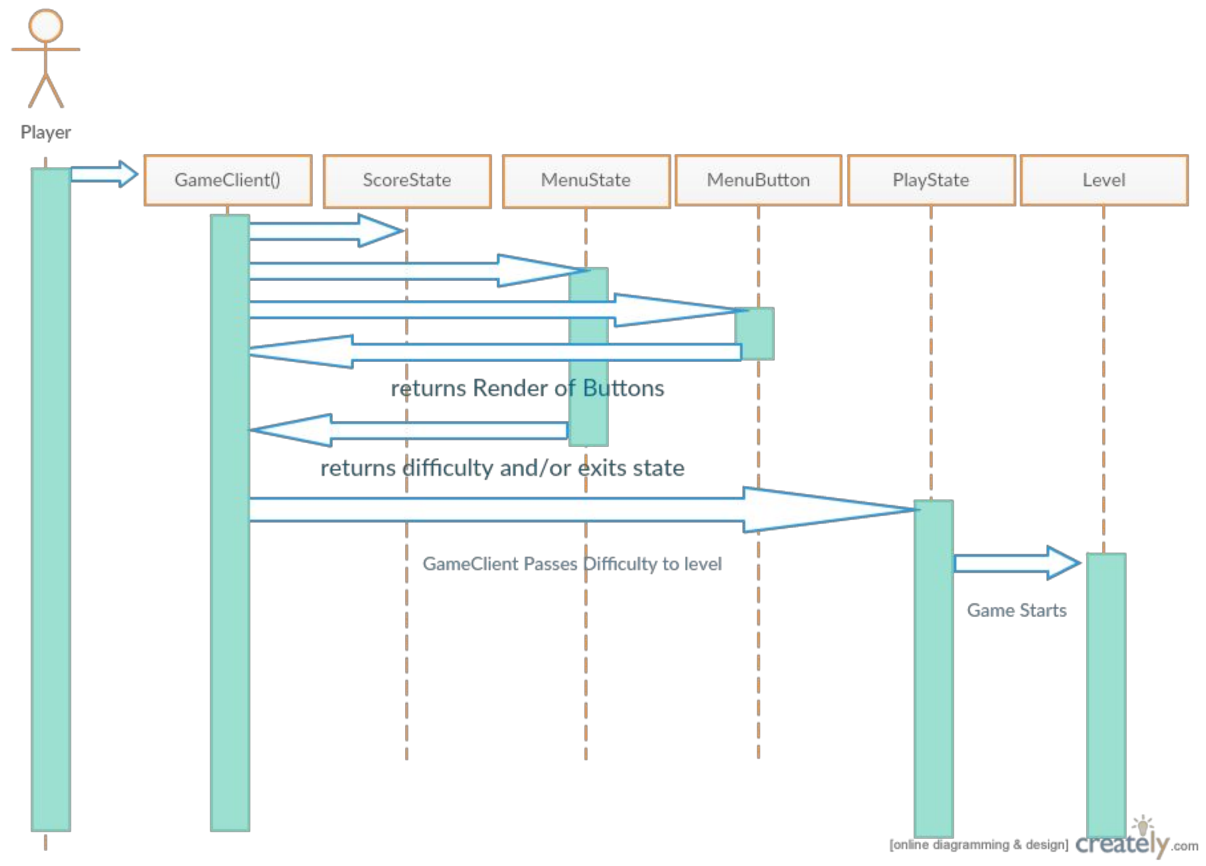
User Interactions:

There will be three primary interactions between the user and this product:

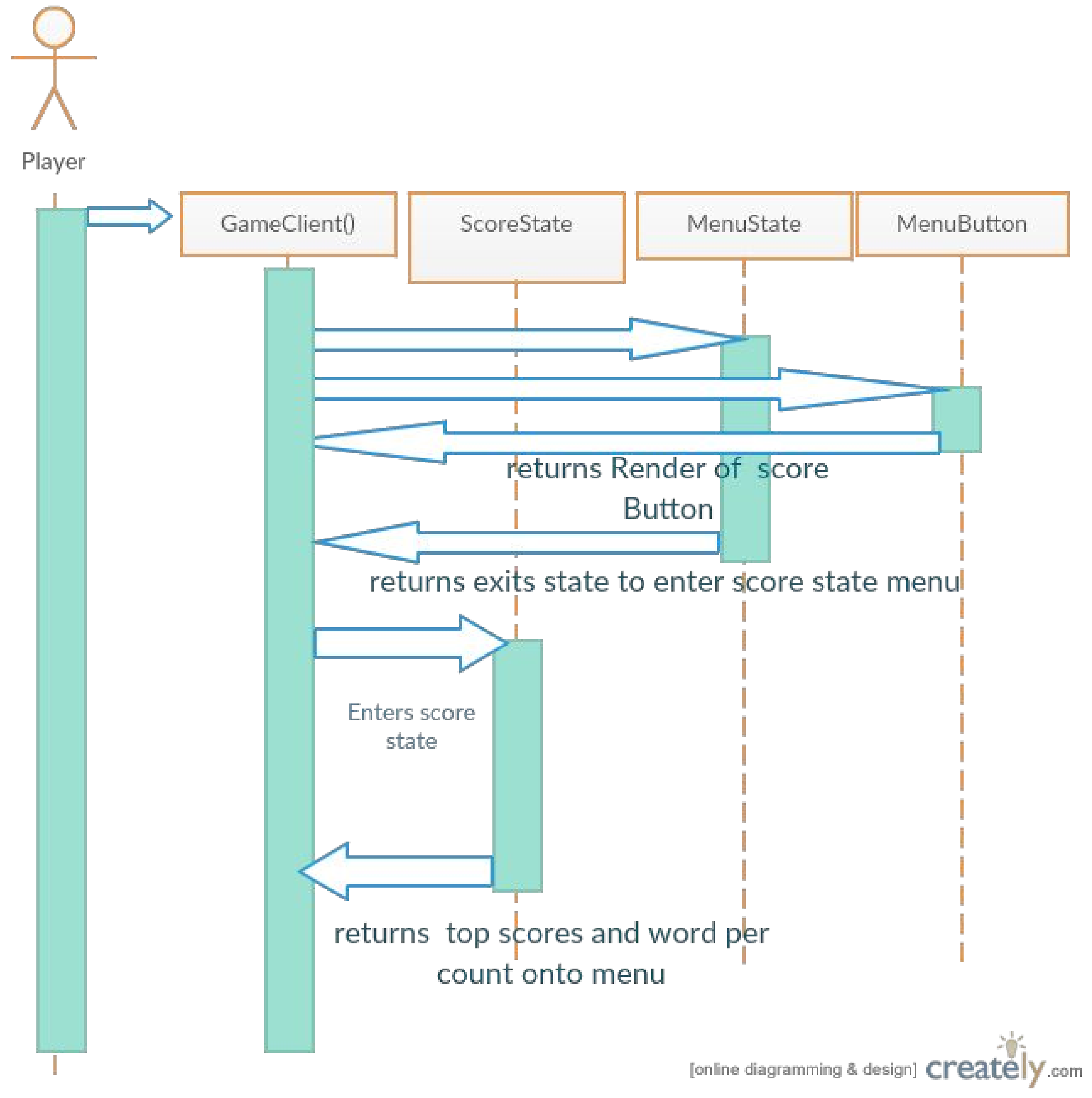
- Playing the game


Player
GameClient()
ScoreState
MenuState
PlayState
Level
Word
WordReader
GUI
InputBox
InputKeys
RenderKey
Calls getState() on all states
Pass - Return dictionary of words
Returns words for rendering
Calls Render and Update
InputKeys calls Render()
RenderKey Renders Returns Keys rendered
InputKeys returns Update()
Returns InputBox
Returns GUI Rendering or Updating
Passes Score and/or exit to State
Exits State
[online diagramming & design] creately.com

- Changing the difficulty from easy, medium, or hard:

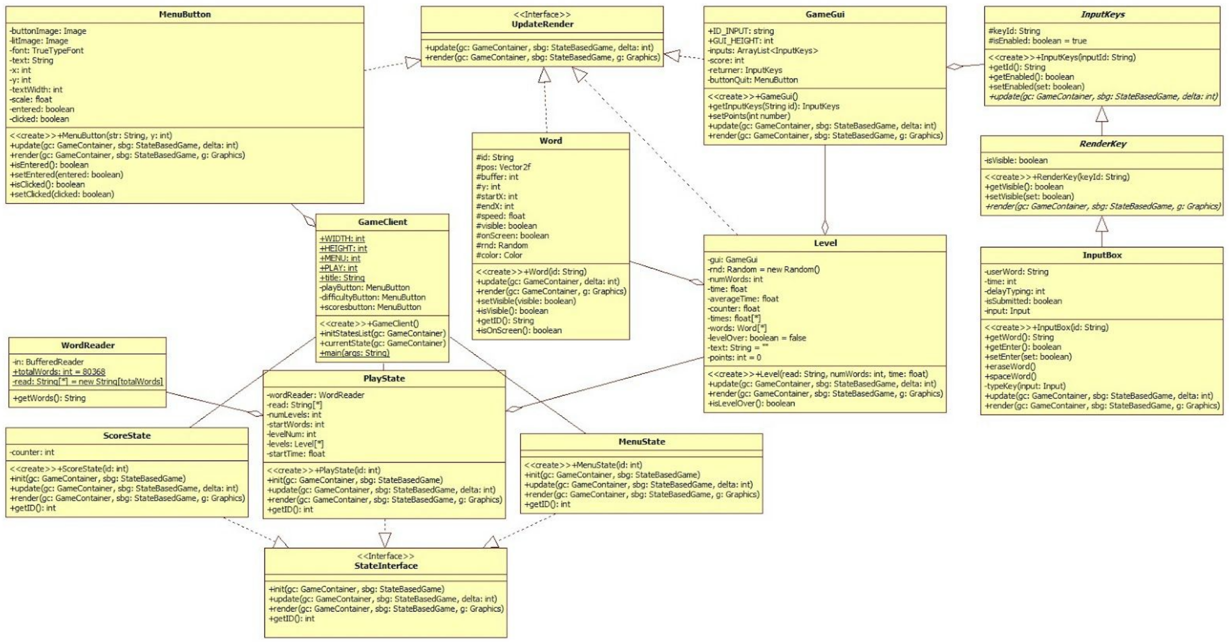

Player
GameClient()
ScoreState
MenuState
MenuButton
PlayState
Level
returns Render of Buttons
returns difficulty and/or exits state
GameClient Passes Difficulty to level
Game Starts
[online diagramming & design] creately.com

- Checking the high scores / words per minute


Player
GameClient()
ScoreState
MenuState
MenuButton
returns Render of score Button
returns exits state to enter score state menu
Enters score state
returns top scores and word per count onto menu
[online diagramming & design] creately.com

Class Diagrams:

As of now, this is our current UML Class diagram. However, We plan to jump back in our Agile stage to change a few things later. For example, if there are too many associations with the GameClient and the State classes, we may implement an abstract class along with the interface. This will add an abstraction layer, and fix the strong associations.


MenuButton
-buttonImage: Image
-litImage: Image
-font: TrueTypeFont
-text: String
-x: int
-y: int
-textWidth: int
-scale: float
-entered: boolean
-clicked: boolean
<<create>>+MenuButton(str: String, y: int)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
+isEntered(): boolean
+setEntered(entered: boolean)
+isClicked(): boolean
+setClicked(clicked: boolean)
<<Interface>>
UpdateRender
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
GameGui
+ID_INPUT: string
+GUI_HEIGHT: int
-inputs: ArrayList<InputKeys>
-score: int
-returner: InputKeys
-buttonQuit: MenuButton
<<create>>+GameGui()
+getInputKeys(String id): InputKeys
+setPoints(int number)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
InputKeys
#keyId: String
#isEnabled: boolean = true
<<create>>+InputKeys(inputId: String)
+getId(): String
+getEnabled(): boolean
+setEnabled(set: boolean)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
RenderKey
-isVisible: boolean
<<create>>+RenderKey(keyId: String)
+getVisible(): boolean
+setVisible(set: boolean)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
Word
#id: String
#pos: Vector2f
#buffer: int
#y: int
#startX: int
#endX: int
#speed: float
#visible: boolean
#onScreen: boolean
#rnd: Random
#color: Color
<<create>>+Word(id: String)
+update(gc: GameContainer, delta: int)
+render(gc: GameContainer, g: Graphics)
+setVisible(visible: boolean)
+isVisible(): boolean
+getID(): String
+isOnScreen(): boolean
GameClient
+WIDTH: int
+HEIGHT: int
+MENU: int
+PLAY: int
+title: String
-playButton: MenuButton
-difficultyButton: MenuButton
+scoresbutton: MenuButton
<<create>>+GameClient()
+initStatesList(gc: GameContainer)
+currentState(gc: GameContainer)
+main(args: String)
Level
-gui: GameGui
-rnd: Random = new Random()
-numWords: int
-time: float
-averageTime: float
-counter: float
-times: float[*]
-words: Word[*]
-levelOver: boolean = false
-text: String = ""
-points: int = 0
<<create>>+Level(read: String, numWords: int, time: float)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
+isLevelOver(): boolean
InputBox
-userWord: String
-time: int
-delayTyping: int
-isSubmitted: boolean
-input: Input
<<create>>+InputBox(id: String)
+getWord(): String
+getEnter(): boolean
+setEnter(set: boolean)
+eraseWord()
+spaceWord()
-typeKey(input: Input)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
WordReader
-in: BufferedReader
+totalWords: int = 80368
-read: String[*] = new String[totalWords]
+getWords(): String
PlayState
-wordReader: WordReader
-read: String[*]
-numLevels: int
-startWords: int
-levelNum: int
-levels: Level[*]
-startTime: float
<<create>>+PlayState(id: int)
+init(gc: GameContainer, sbg: StateBasedGame)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
+getID(): int
ScoreState
-counter: int
<<create>>+ScoreState(id: int)
+init(gc: GameContainer, sbg: StateBasedGame)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
+getID(): int
MenuState
<<create>>+MenuState(id: int)
+init(gc: GameContainer, sbg: StateBasedGame)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
+getID(): int
<<Interface>>
StateInterface
+init(gc: GameContainer, sbg: StateBasedGame)
+update(gc: GameContainer, sbg: StateBasedGame, delta: int)
+render(gc: GameContainer, sbg: StateBasedGame, g: Graphics)
+getID(): int