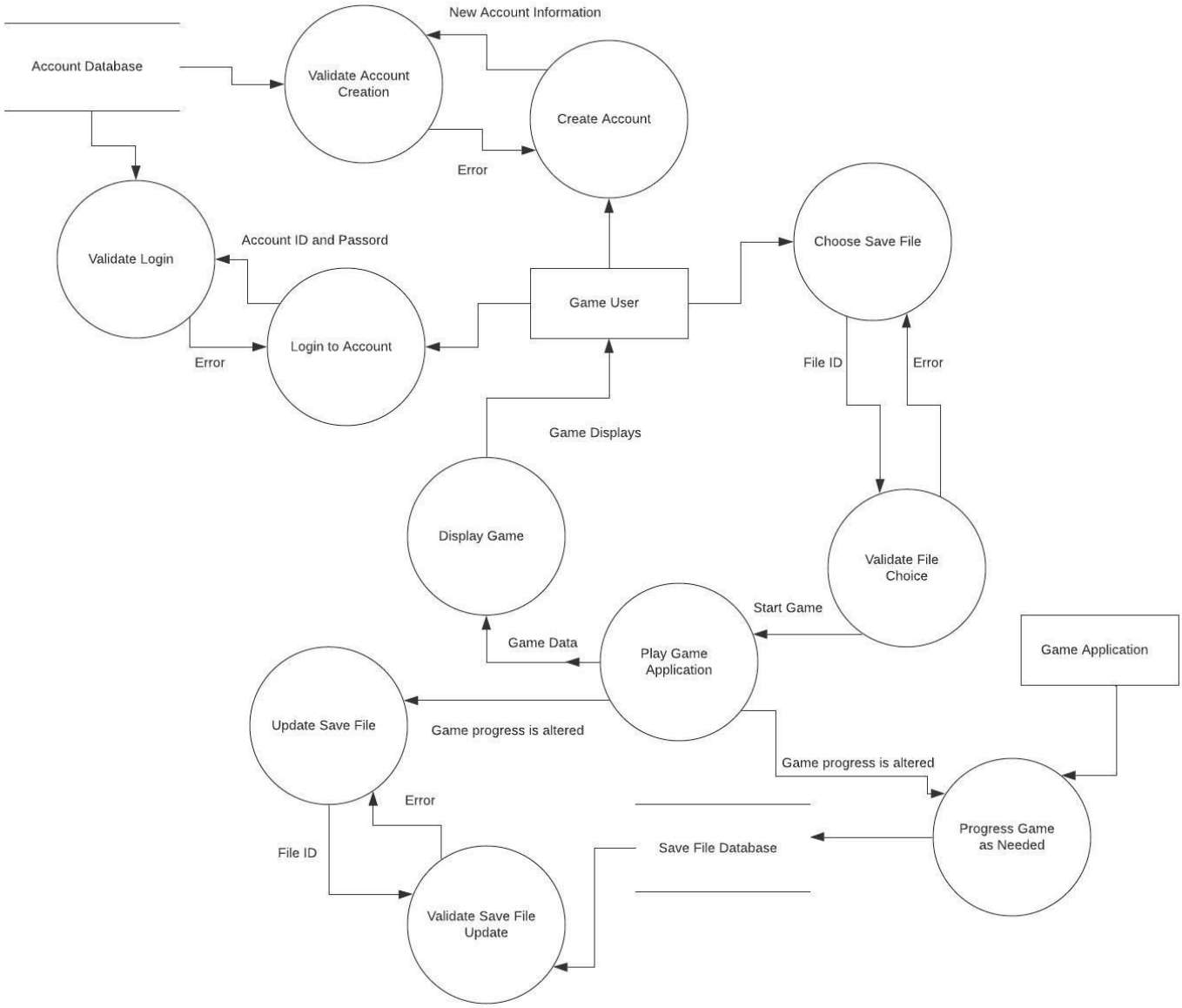
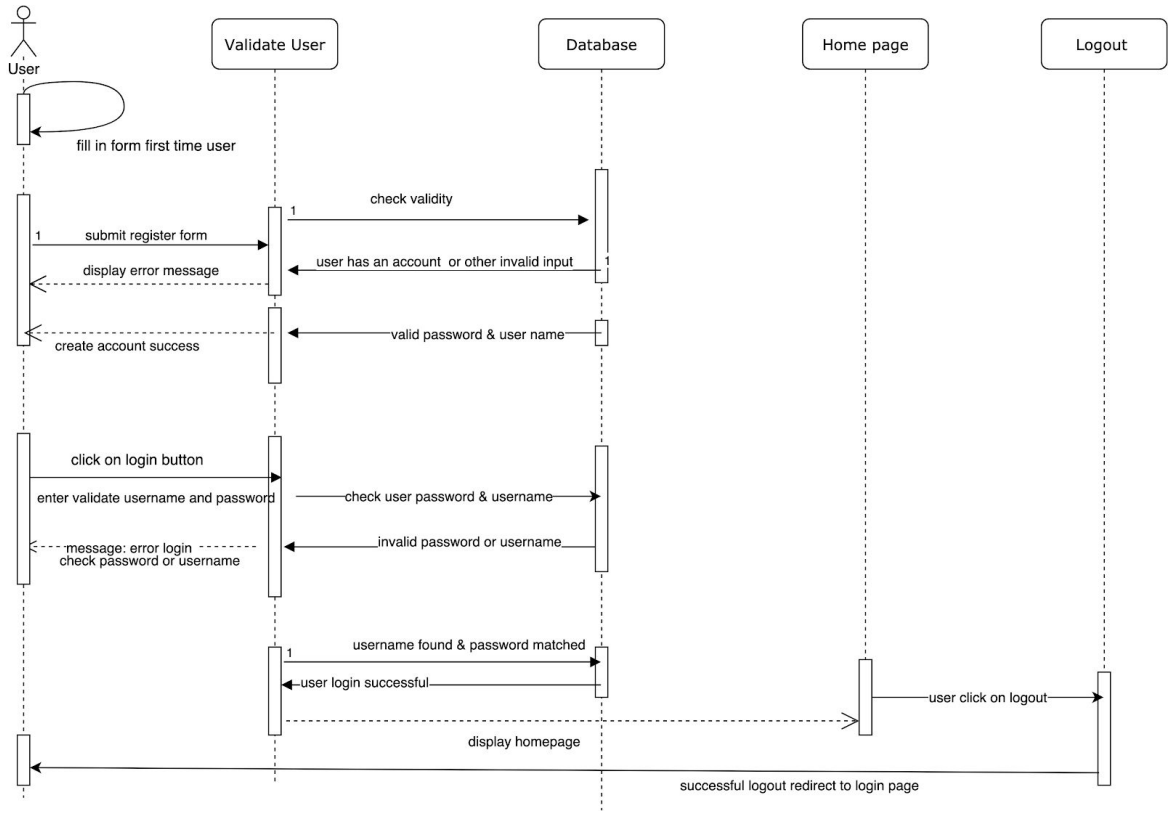
Group 1
Brian Albert Redoloza
Dhirtitapa Ray
Uyen Nguyen
Yenni Lam

Data Flow Diagram

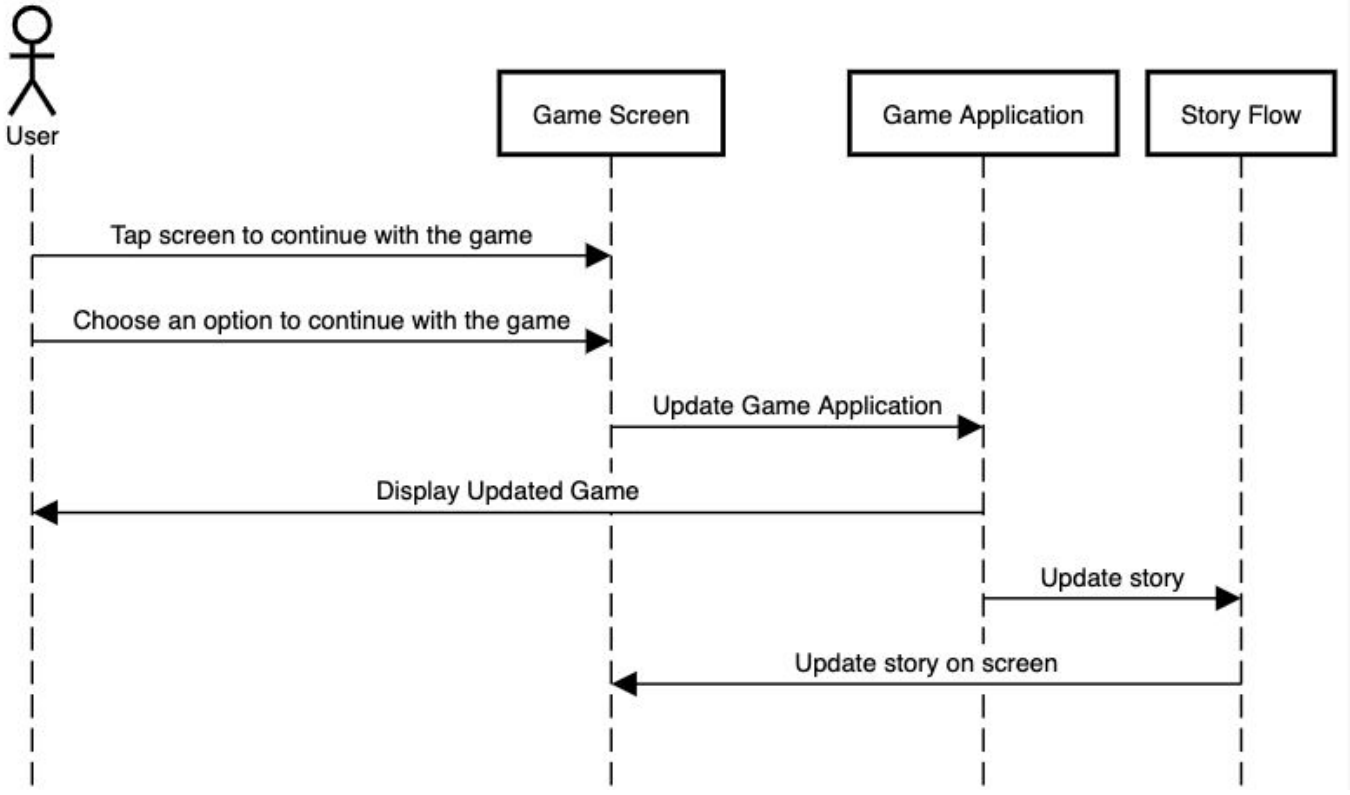


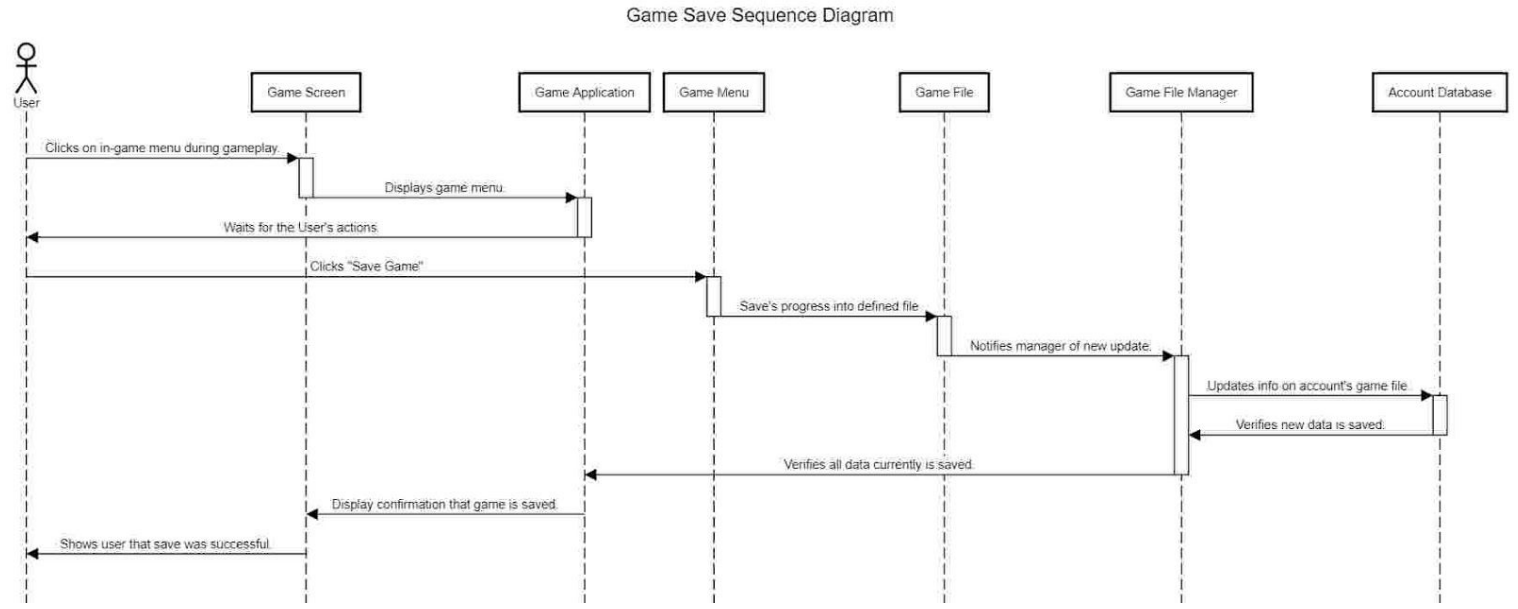
Sequence Diagrams

Login/Logout Sequence Diagram

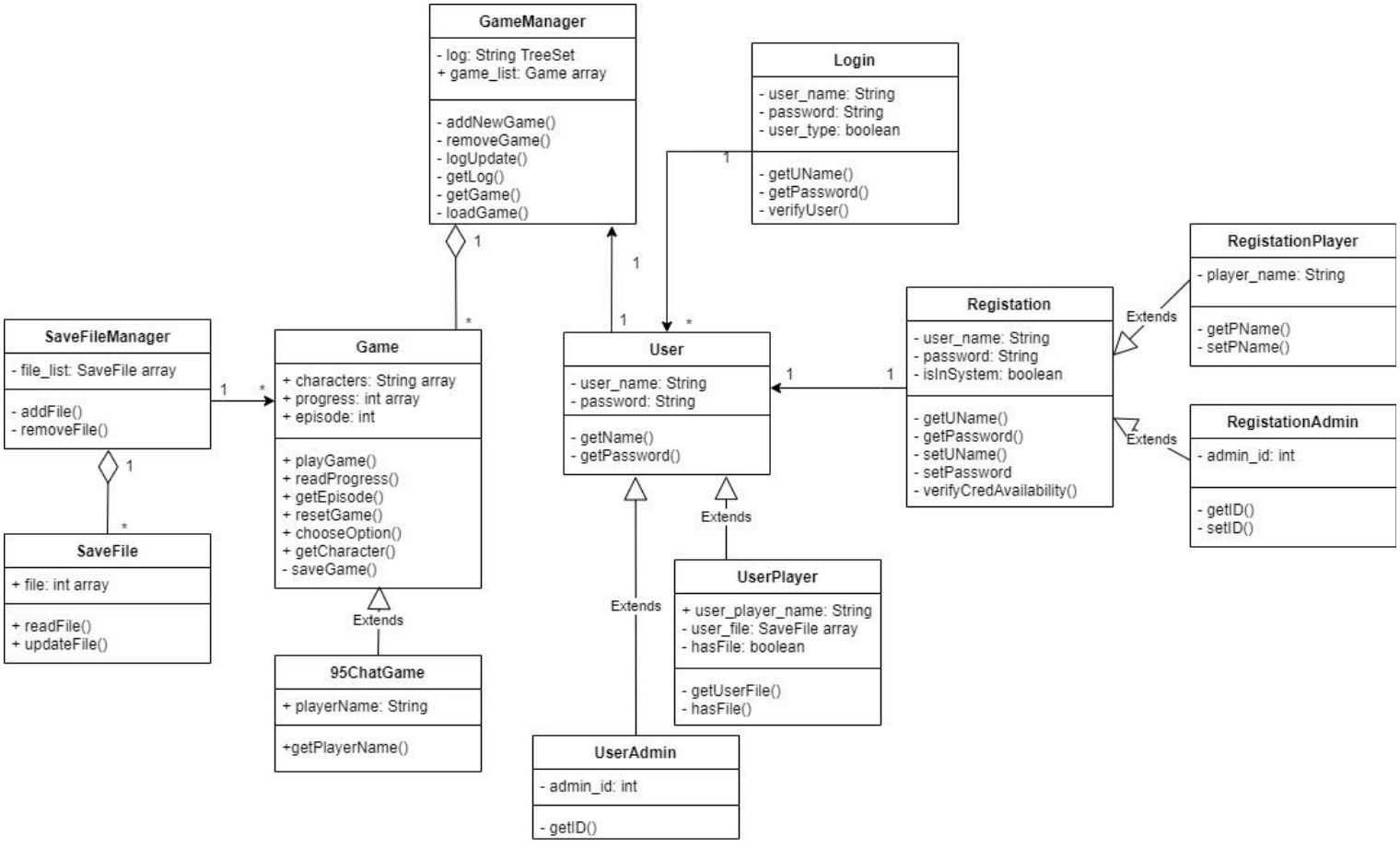


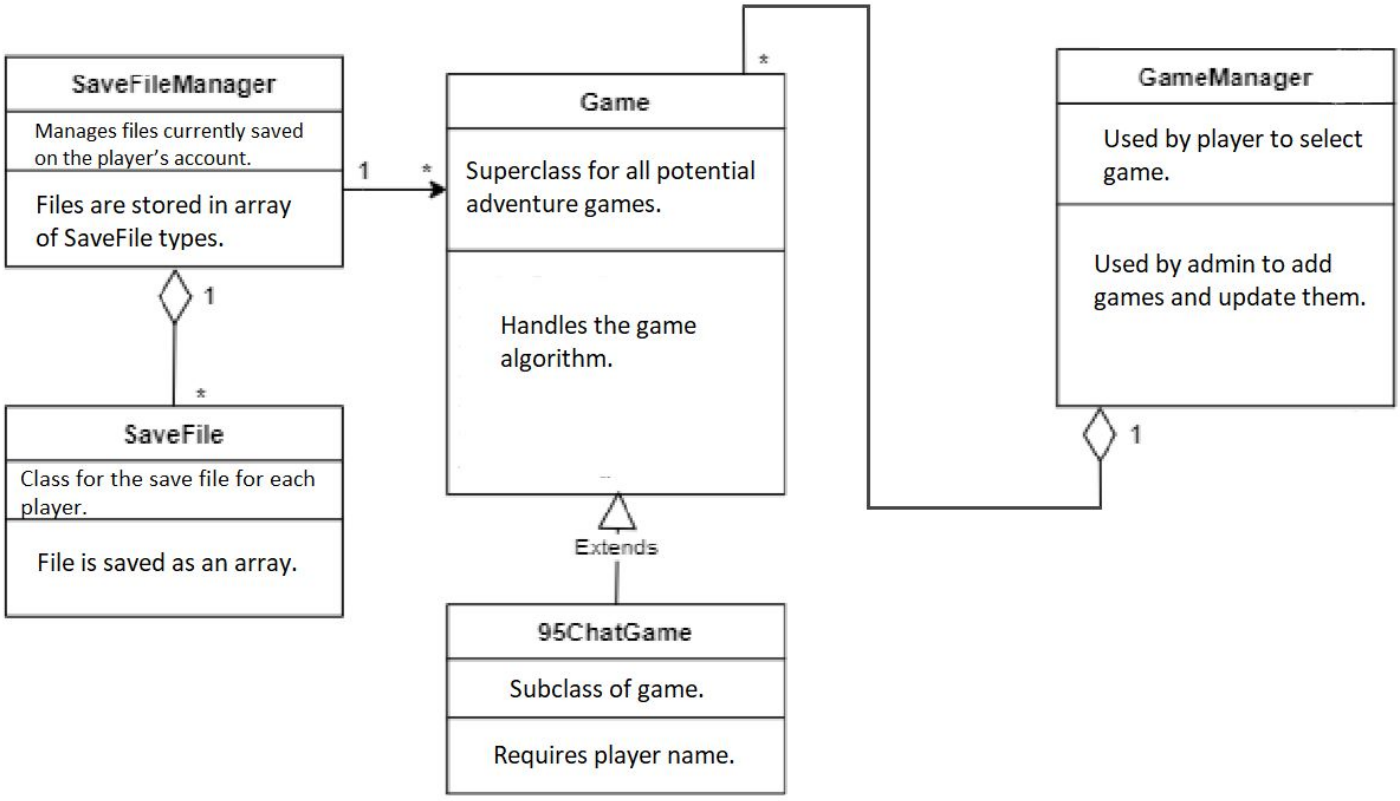
Play Game Sequence Diagram

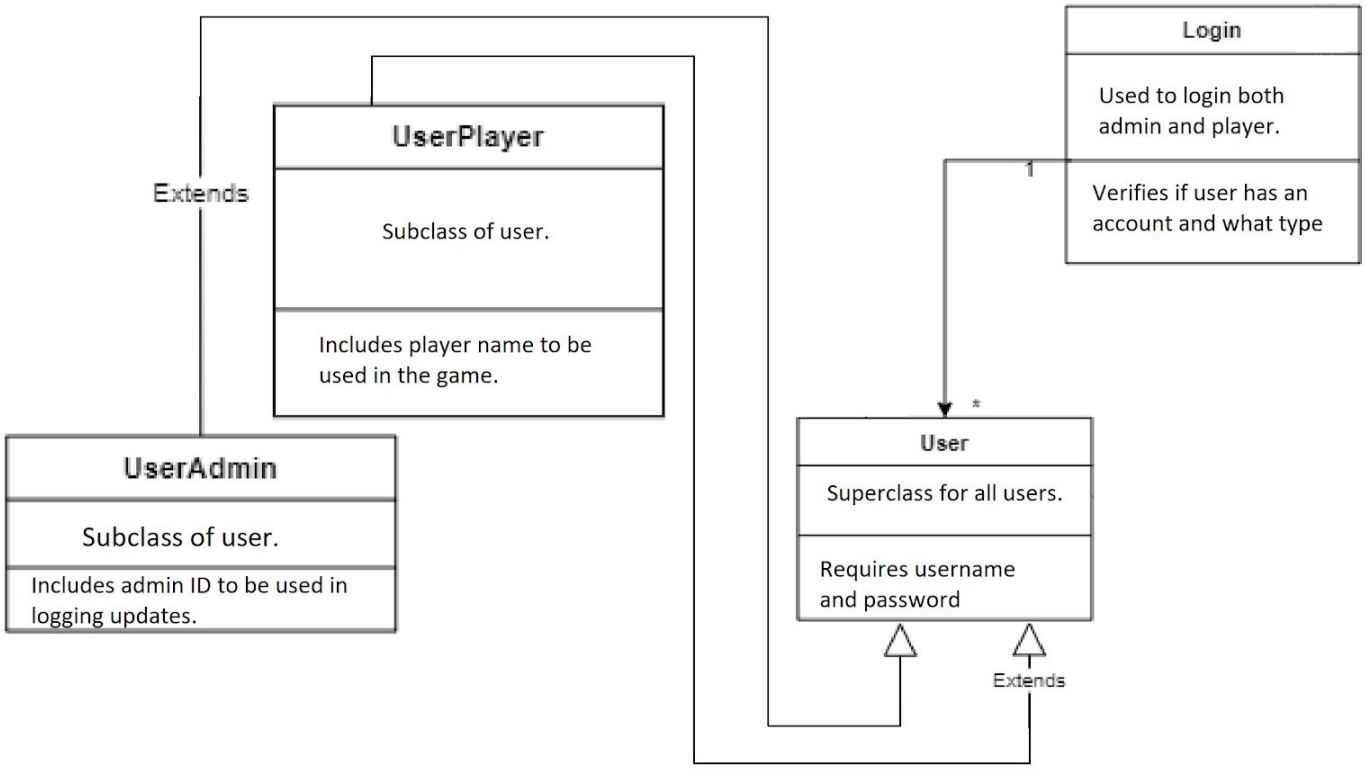


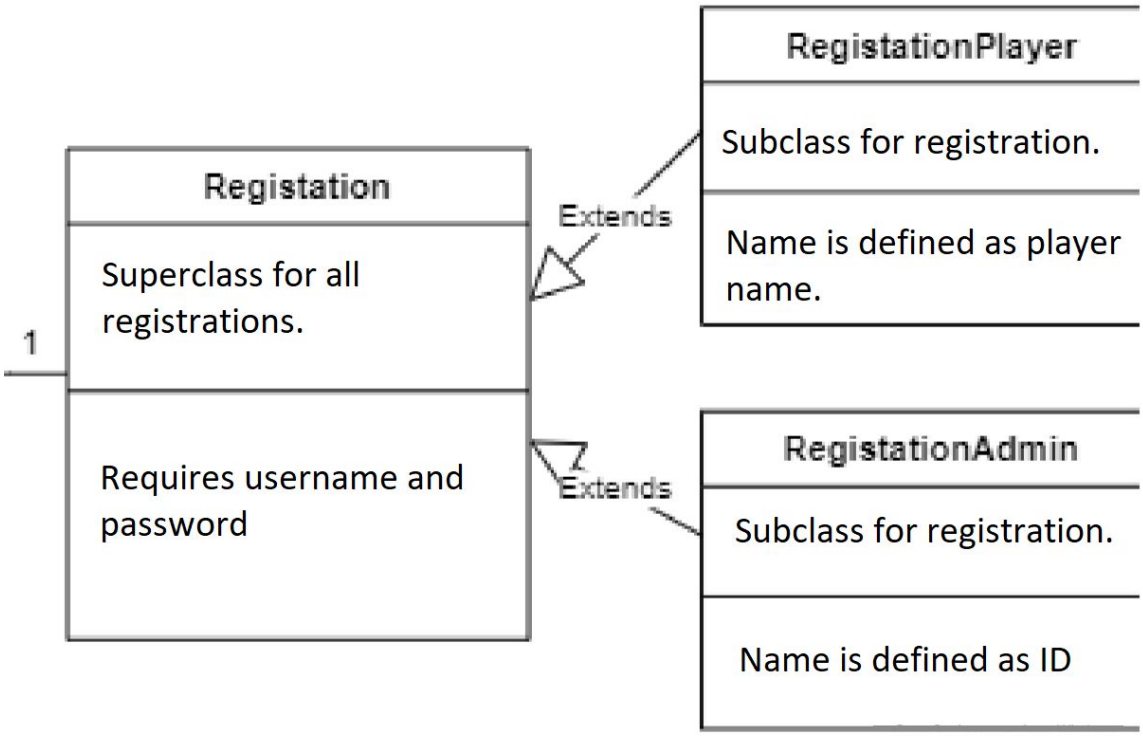


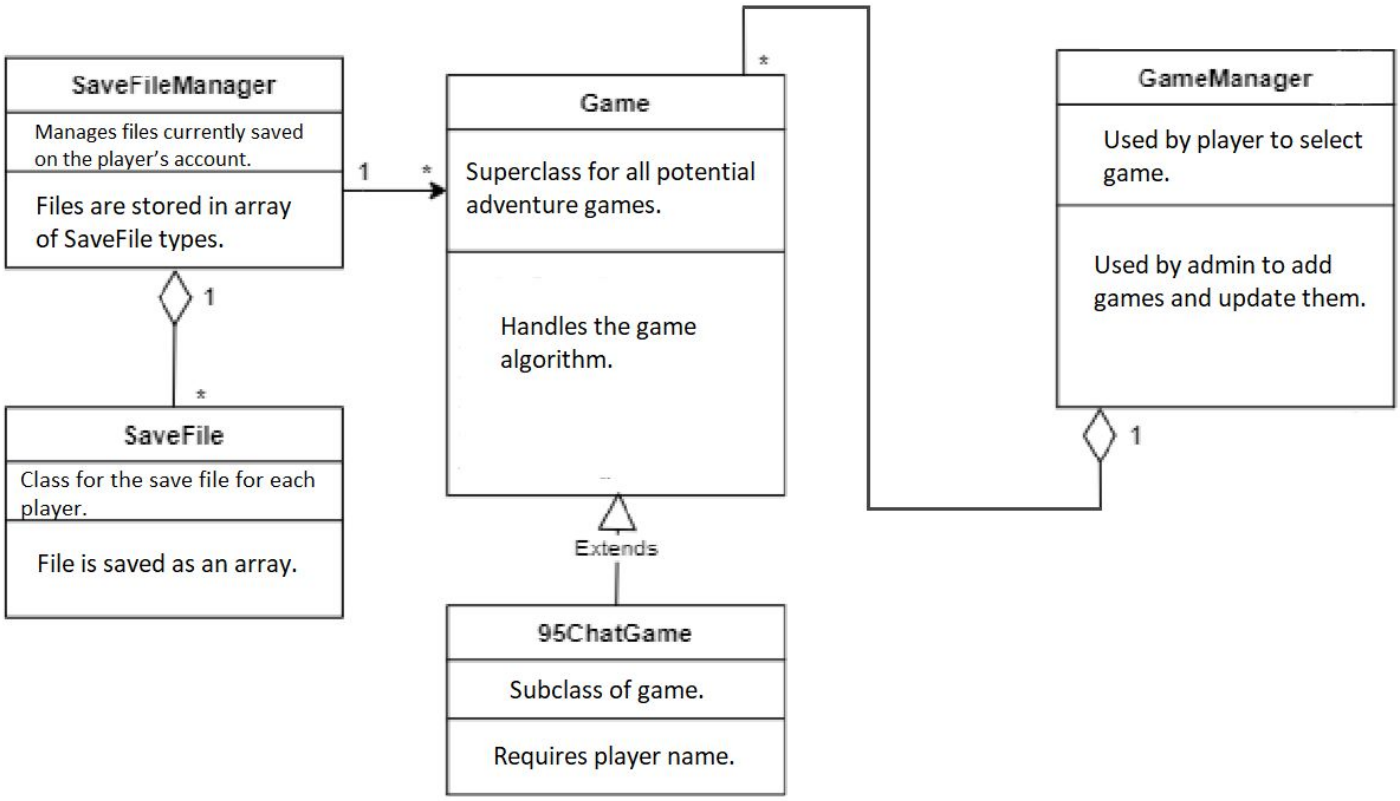
Class Diagram



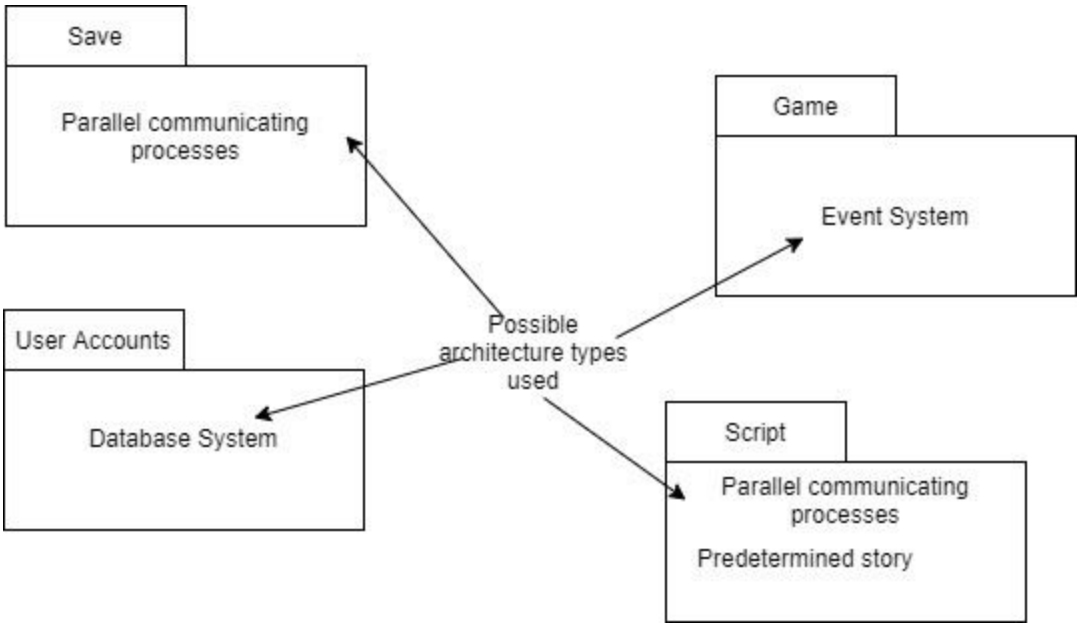

SaveFileManager
Manages files currently saved on the player's account.
Files are stored in array of SaveFile types.
1
*
1
*
SaveFile
Class for the save file for each player.
File is saved as an array.
Game
Superclass for all potential adventure games.
Handles the game algorithm.
*
Extends
95ChatGame
Subclass of game.
Requires player name.
GameManager
Used by player to select game.
Used by admin to add games and update them.
1


Extends
UserPlayer
Subclass of user.
Includes player name to be used in the game.
UserAdmin
Subclass of user.
Includes admin ID to be used in logging updates.
Login
Used to login both admin and player.
Verifies if user has an account and what type
1
*
User
Superclass for all users.
Requires username and password
Extends


Registation
Superclass for all registrations.
Requires username and password
1
Extends
RegistationPlayer
Subclass for registration.
Name is defined as player name.
Extends
RegistationAdmin
Subclass for registration.
Name is defined as ID


SaveFileManager
Manages files currently saved on the player's account.
Files are stored in array of SaveFile types.
1
*
1
*
SaveFile
Class for the save file for each player.
File is saved as an array.
Game
Superclass for all potential adventure games.
Handles the game algorithm.
*
Extends
95ChatGame
Subclass of game.
Requires player name.
GameManager
Used by player to select game.
Used by admin to add games and update them.
1

Multi-level architecture



Script Preview (subject to change):

Pat: ARE YOU SERIOUS?
Pat: answer me, [insert name]!
Pat: HEY!
Pat: [insert name]!
Pat: asdopabpaewagalsdkjnbfjoiij
Pat: You can't do this to me!
Pat: Tell me this isn't true. You actually replaced me?

- Option 1: "Yes. He asked me because you obviously need time off."
- Option2: : "Yes, he asked me about that, but maybe we should focus more on getting our doors unlocked."

Pat: Gosh, am I ever good enough for this company?
Pat: This is what it comes down to?
Pat: What have I done wrong, [insert name]?
Pat: I did my damn har har hard ess t.
Pat: I dd ev erything he a sked me, an yet ths happpens … wh a why...

- Option 1: "Calm down. Spamming me isn't going to get anything done. You're only proving Leader's point by doing this"
- Option 2: "Slow down. I'm sorry, Pat. Talk to me. What exactly is going on?"

Suddenly Pat stops responding.

- Option Null: "Pat? Where are you?"'
- Option Null: "Pat? Are you still there?"'

Pseudocode for Game:
variables:

- aCount - int
- bCount - int
- nullCount - int
- episodes - array
- gameScript - array
- playerScriptA - array
- playerScriptB - array
- gameEndingA - array
- gameEndingB - array
- playerEndingA - array
- playerEndingB - array
-

methods:

- getACounts() - returns current aCount
- getBCounts() - returns current bCount
- getNullCounts() - returns current nullCount
- getEpisode() - returns current episode
- getResults() - returns the ending.
- getGScript() - returns the current script of the game.
- getPScript() - returns the current script of the game on the player's side.
- countOptions() - determines game results
- getResults() - returns the ending of the game.
- playGame() - starts game and continues until it stops

```
// to determine ending and debug
getACounts() {
        return aCount;
}
// to determine ending and debug
getBCounts() {
        return bCount;
}

// to address same answers and for debugging
getNullCounts() {
        return nullCount;
}

// to be used for potential save file and debugging
getEpisode(int i) {
        return episodes at i
}

// to get the current line of the game's script
getGScript(int i) {
        returns gameScript at i.
```

```
}
// to get the current line of the player's script depending on what they choose
getPScript(array pscript, int i) {
	returns the given pscript (playerScriptA or playerScriptB) at i.
}

// used to determine how the story will end.
countGame() {
	If (player chooses an A option) {
		aCount++
	}
	Else if (player chooses a B option) {
		bCount++
	}
	Else {
		nullCount++
	}
	Iterate through playerScriptA and playerScriptB.
}

getResults() {
	if (aCount > bCount) {
		Iterate through gameEnding
		Iterate through endingA
	}
	else {
		clterate through gameEndingA
		Iterate through endingB
	}
	Return "the end" sequence //either a play again or refresh back to a different page
}

playGame() {
	while player has not exited or game ending has not been reached {
		Iterate through all the scripts and run the necessary functions.
		Exit when player exits game or has reached the last index of their chosen ending
		// playerEndingA or playerEndingB
	}
}
```