CPSC 2720 FALL

Youth.txt

Altair

Aldrin Azucena, William Hughson, Brian Cruz

November 17, 2019

Table Of Contents

Introduction

Synopsis

"**Whoosh* You are a young man/woman in highschool. The modern world has been hard on you and you have had to transfer into a new school on your last year of highschool. This is your last chance to make the most of your precious youth before it is all gone in 20XX. Get good grades, make friends, and find romance the sky is the limit. However anything can happen at school and your choices may lead to a fulfilling life or a bleak future.*"

That is our summary story of our game called Youth.txt. It is a slice of life-esque, role-playing, dialogue-based game that involves social simulation scenarios throughout each day(turn) in order for the created character to become the 'perfect' student against time management. It also incorporates a touch of dating simulation, trivia, and non-fictional lifestyle as the game is set on a modern setting, adding the real-life atmosphere to the game.

Rules & Guidelines

- The goal of this game is to maximise the stats of the user-created student, gain more relationships, and accomplish many tasks in the game before they graduate.
- The max level of stats is 60, and will be determined by status name. To increase the stats' level, a player must find events and opportunities in the environment to increase the level, or make the right choice to increase them. If a user fails an opportunity, event, or makes a wrong choice, the stats will not change. So meaning that any stats can either increase, or not be modified. Each stat will be increased by 1 only.
- The duration of the game will go for 6 days, this game is turn-based meaning each day will be a turn for the user to make a move, and the time stages will goes from: Morning, Lunch, Afternoon, After School, and Evening within each days.
- There is a helper guide named Alex Caruso with a gender determined by the user will be used for tutorials & help guide on how to play the game, and the tips & tricks to get advice increasing stats, or hinting on making the right choices in dialogue.
- There will be also Affection status level that will be increased by interacting with students (NPCs) from the events. Affection level will be increased when the user makes right dialogue choices with the students the user interacts. The affection level will be increased by +6^n since there are 6 levels: Acquaintance, Colleague, Friend, Best Friend, & Genuine Friend/Romantic Partner. Depending on the choices the user made, the affection status may either increase, stay as it is, or decreased. The max modifier for best answer is 2, the worst answer is -6^2.
- Puzzles in this game will be based on the questions asked on pop quizzes and exams. They can be figuring out which is the best dialogue choices when interacting with objects, environment and students in the game. The biggest challenge in the game is 'Time Management' as the user needs to make the right decisions to choose on what they should do before continuing the next day (turn).

- There will be an options menu that will include status overview, relationships overview.The user can save their game progression at the end of the day, but not during the dialogue, or making major dialogue choices.

Outline

The following proposal will outline critical infrastructures associated with team Altair. We will describe potential risks while assembling of Youth.txt. Furthermore, this document will clearly outline each team member's role. Proceeding our project management section team Altair will illustrate our development process. This section will describe all aspects concerning the construction of Youth.txt. We will demonstrate the subsections of Youth.txt in the form of a Unified Modeling Language class and sequence diagrams.

Project Management

Team Altair has been instructed to meet specific system requirements in the construction of Youth.txt. For optimal development of the project, Altair has designated each member into a role that amplifies their strengths. Aldrin Azucena at team leads because of is always on top of its attitude. Brian Cruz at design lead because of his extensive knowledge of object-oriented principles. William Hughson at Quality Assurance because of his eagerness to deliver a bug-free product. Furthermore, team Altair has made precautions to cope with potential risks that can plague Youth.txt. Below are two subsections, team organization and risk management, that will go into greater detail about team Altair's project management.

Team Organization

Altair has come together to exceed the expectations. Being successful in any project starts with being successful as a team. Our team organization is designed for optimal productivity and success rate. With this design structure in our minds, we delegated Aldrin Azucena as our team lead. Aldrin was the clear choice for our team lead. Aldrin first introduced the concept of our game. He also shows critical qualities that are demanded by a team lead. Aldrin makes sure all of us are on the same page with our tasks. He makes sure our team meetings stay on topic and ensures all deadlines are met. Unfortunately, having only a good team leader does not lead to the work Altair demands. While working in a team, there will be several different coding styles being implemented. For this reason, our team has established a design lead. For the Youth.txt project, Brain Cruz will be our design lead. Brain has the most experience with object-oriented techniques and will enforce those principles while the construction of Youth.txt. Altair's team organization would be complete if software bugs did not exist. Therefore, we assigned William Hughson to quality assurance. All members are new with writing proper unit tests. However, every member is currently gaining experience with all thing's unit tests. Billy has shown a keen eye in the choosing and writing of unit tests. Each member in Altair is in the specific role to lead to the most success for Youth.txt.

Risk Management

With the development of any projects come uncertainties. While developing Youth.txt, Altair understands that there is an assortment of risk associated with the construction of Youth.txt. Altair recognizes the potential dangers of time management, inexperience with projects of this scale and or working in groups, background knowledge in C++ and the inability for team members to produce. However, Altair has developed possible solutions for each of these risks detailed below.

Teams must have the ability to meet deadlines and complete subtasks at a predetermined point of time. Without this skill set, teams become massively behind schedule, periods do not become met, and clients can become frustrated. Altair solution for this risk is with are reliable team lead Aldrin Azucena. As stated before, Aldrin always keeps everyone up to date on what and when deadlines need to be met. Team Altair is also in constant communication with each other through a program called Slack, which will be explained later.

Altair is wholly composed of post-secondary students. Therefore, our experience with big scale projects is limited. Furthermore, Altair is a newly formed group. The risk associated with freshly constructed groups consists of clashing personalities, poor communication and decision making. Even though our experience is limited, we are eager to efficiently and effectively tackle the system requirements. Altair is very communication heavy. We have assembled channels for providing and discussion new and current ideas about Youth.txt. Furthermore, we at Altair strive for our professionalism. This means leaving our personal feelings out of the way of hindering the project.

As stated before, Altair is composed of post-secondary students. Therefore, our experience is lacking in specific categories. One of these categories could potentially be that one of our members is inexperienced with C++ for the scale of this project. Altair believes in helping each other through every step of the way. If one member gets stuck, said member goes to a channel, and we will all brainstorm together to try and fix the problem.

Development Process

While constructing Youth.txt team, Altair will need proper development structure to be successful. For this reason, we have established a system that each team member will follow throughout this project. Each member will have their working branch. Once a team member confidently feels that they have completed a subtask, they will push their changes to a rough draft branch. The rough draft branch is where Youth.txt is bombarded with rigours testing methods. After team Altair has exterminated all the bugs, the team lead will push the changes to the master branch. The master branch will be a clean copy of our project. Only the team lead can pull, push or alter the master branch. This will ensure the safety of our final product.

Team Altair, as stated before, will be using a messaging software known as Slack. Slack is a fantastic assist to any team. This will allow us to communicate with each other anytime and anywhere. Furthermore, Slack allows team Altair the ability to have custom channels associated with our project. Our slack channels currently consist of idea-brainstorming, design-milestone, general and text-based-adventure-game. This will ensure that any issues that arise while developing Youth.txt are adequately categorized and efficiently handled.

While on the subject of channels, team Altair will create a unique channel concerned with bugs. This channel associated with our rough draft repository will help the detection and termination of any flaws obstructing the system. Furthermore, because of our development structure, every member will be testing, detecting, reporting, and resolving all bug related aspects. However, team members that do detect issues will our system will be required to report the problem to the bug channel as well as the rough draft repository.

Software Design

Design

In our design we have Manager classes which have specific functionality or maintain a portion of the game. An example is the CharacterManager which keeps all our named characters and maintain their data. GameStateManager is where the main program starts and ends depending on whether the Player succeeds through the school days. GameStateManager is also the central area where most of the managers are held with the exception of Player and CharacterManager. Our saving and loading of the game will be handled by GameDataManager which collects the data together and then sends it to FileManager which will either write or read data we want to persist, such as the player and characters, into or from text files.We have the Player class, that extends from Person, which will hold information for the player such as his/her name, gender and stats, which is a seperate class that is aggregated to Player. We have the event class which will hold an introduction scene and choices which will affect the Player's stats or a Character's affection. We have a StringMagician class whose only purpose is to change strings for input correction or making output fancy.

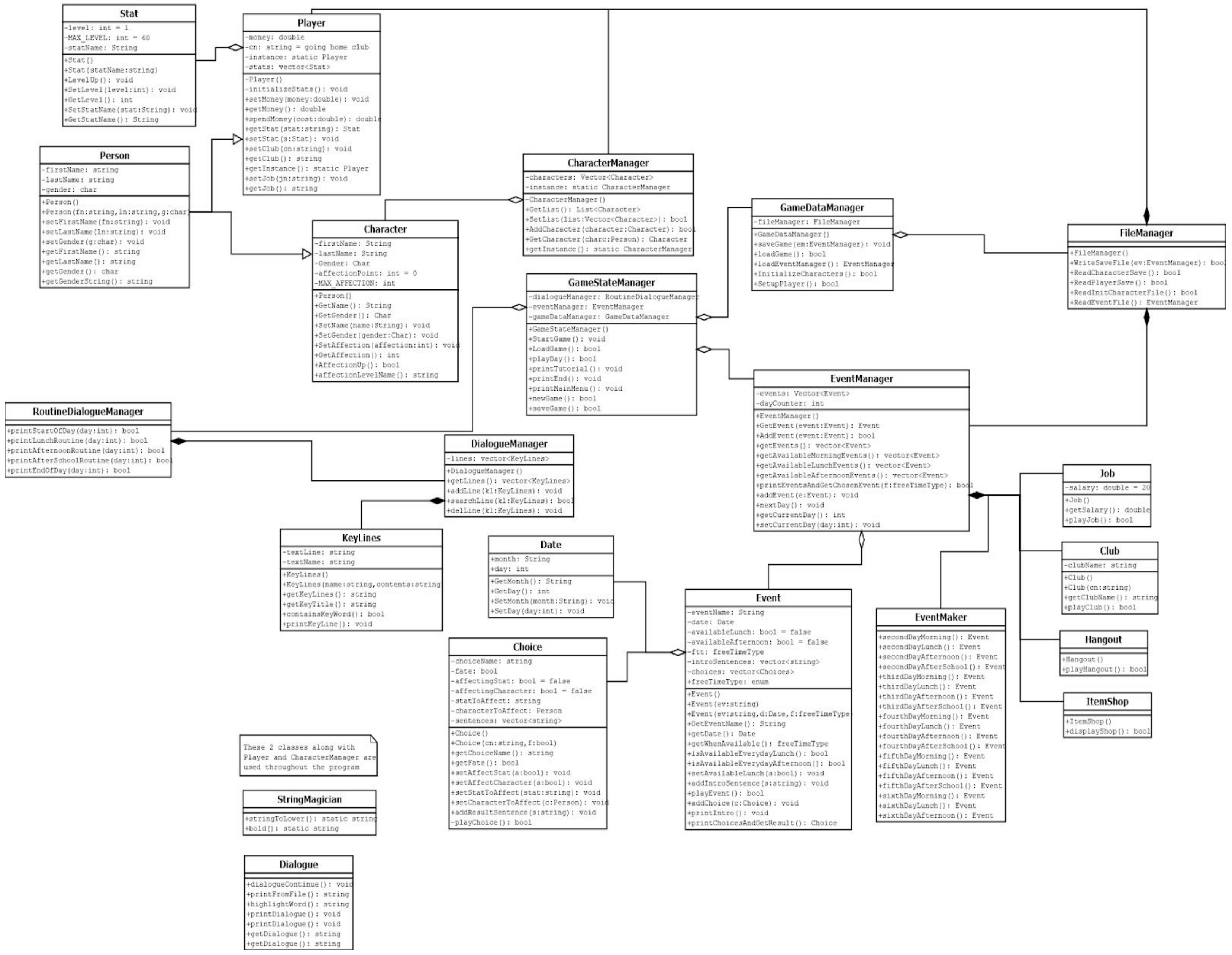
Design Rationale

Player and CharacterManager use the singleton pattern on the basis that these classes (The Character objects inside CharacterManager) should be a one instance object during run-time and also their resources need to be accessed and changed throughout the program. We have Player and Character extend from the Person class as Player and Character share common attributes (first name, last name, and gender) and the purpose of the Person class was to encapsulate said common data. We also have classes such as StringMagician and DialogueManager which delegates some responsibilities such as string manipulation to them rather than say in RoutineDialogueManager which will output repetitive text to the user. Back in our previous design we had Job, Hangout, and Club inherit from Event. They no longer inherit from Event on the reason that they are constant events and would require complex instantiation if inheriting from event and to be found easily by our program.

Appendices

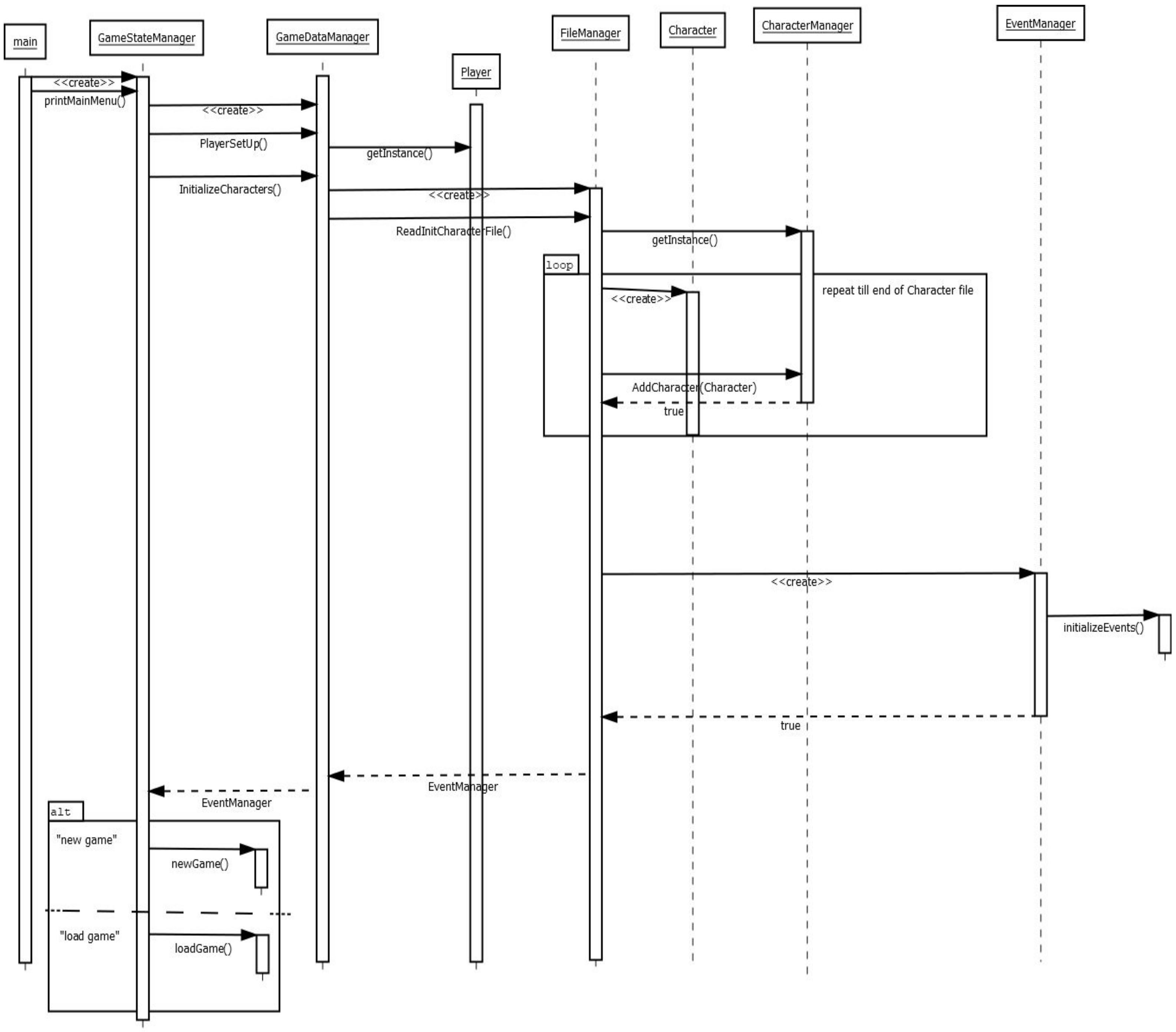
Appendix A

Class Diagram


Stat
-level: int = 1
-MAX_LEVEL: int = 60
-statName: String
+Stat()
+Stat(statName:string)
+LevelUp(): void
+SetLevel(level:int): void
+GetLevel(): int
+SetStatName(stat:String): void
+GetStatName(): String
Player
-money: double
-cn: string = going home club
-instance: static Player
-stats: vector<Stat>
-Player()
-initializeStats(): void
+setMoney(money:double): void
+getMoney(): double
+spendMoney(cost:double): double
+getStat(stat:string): Stat
+setStat(s:Stat): void
+setClub(cn:string): void
+getClub(): string
+getInstance(): static Player
+setJob(jn:string): void
+getJob(): string
Person
-firstName: string
-lastName: string
-gender: char
+Person()
+Person(fn:string,ln:string,g:char)
+setFirstName(fn:string): void
+setLastName(ln:string): void
+setGender(g:char): void
+getFirstName(): string
+getLastName(): string
+getGender(): char
+getGenderString(): string
Character
-firstName: String
-lastName: String
-Gender: Char
-affectionPoint: int = 0
-MAX_AFFECTION: int
+Person()
+GetName(): String
+GetGender(): Char
+SetName(name:String): void
+SetGender(gender:Char): void
+SetAffection(affection:int): void
+GetAffection(): int
+AffectionUp(): bool
+affectionLevelName(): string
CharacterManager
-characters: Vector<Character>
-instance: static CharacterManager
-CharacterManager()
+GetList(): List<Character>
+SetList(list:Vector<Character>): bool
+AddCharacter(character:Character): bool
+GetCharacter(charc:Person): Character
+getInstance(): static CharacterManager
GameDataManager
-fileManager: FileManager
+GameDataManager()
+saveGame(em:EventManager): void
+loadGame(): bool
+loadEventManager(): EventManager
+InitializeCharacters(): bool
+SetupPlayer(): bool
FileManager
+FileManager()
+WriteSaveFile(ev:EventManager): bool
+ReadCharacterSave(): bool
+ReadPlayerSave(): bool
+ReadInitCharacterFile(): bool
+ReadEventFile(): EventManager
GameStateManager
-dialogueManager: RoutineDialogueManager
-eventManager: EventManager
-gameDataManager: GameDataManager
+GameStateManager()
+StartGame(): void
+LoadGame(): bool
+playDay(): bool
+printTutorial(): void
+printEnd(): void
+printMainMenu(): void
+newGame(): bool
+saveGame(): bool
RoutineDialogueManager
+printStartOfDay(day:int): bool
+printLunchRoutine(day:int): bool
+printAfternoonRoutine(day:int): bool
+printAfterSchoolRoutine(day:int): bool
+printEndOfDay(day:int): bool
DialogueManager
-lines: vector<KeyLines>
+DialogueManager()
+getLines(): vector<KeyLines>
+addLine(kl:KeyLines): void
+searchLine(kl:KeyLines): bool
+delLine(kl:KeyLines): void
EventManager
-events: Vector<Event>
-dayCounter: int
+EventManager()
+GetEvent(event:Event): Event
+AddEvent(event:Event): bool
+getEvents(): vector<Event>
+getAvailableMorningEvents(): vector<Event>
+getAvailableLunchEvents(): vector<Event>
+getAvailableAfternoonEvents(): vector<Event>
+printEventsAndGetChosenEvent(f:freeTimeType): bool
+addEvent(e:Event): void
+nextDay(): void
+getCurrentDay(): int
+setCurrentDay(day:int): void
Job
-salary: double = 20
+Job()
+getSalary(): double
+playJob(): bool
Club
-clubName: string
+Club()
+Club(cn:string)
+getClubName(): string
+playClub(): bool
Hangout
+Hangout()
+playHangout(): bool
ItemShop
+ItemShop()
+displayShop(): bool
KeyLines
-textLine: string
-textName: string
+KeyLines()
+KeyLines(name:string,contents:string)
+getKeyLines(): string
+getKeyTitle(): string
+containsKeyWord(): bool
+printKeyLine(): void
Date
+month: String
+day: int
+GetMonth(): String
+GetDay(): int
+SetMonth(month:String): void
+SetDay(day:int): void
Event
-eventName: String
-date: Date
-availableLunch: bool = false
-availableAfternoon: bool = false
-ftt: freeTimeType
-introSentences: vector<string>
-choices: vector<Choices>
+freeTimeType: enum
+Event()
+Event(ev:string)
+Event(ev:string,d:Date,f:freeTimeType)
+GetEventName(): String
+getDate(): Date
+getWhenAvailable(): freeTimeType
+isAvailableEverydayLunch(): bool
+isAvailableEverydayAfternoon(): bool
+setAvailableLunch(a:bool): void
+addIntroSentence(s:string): void
+playEvent(): bool
+addChoice(c:Choice): void
+printIntro(): void
+printChoicesAndGetResult(): Choice
EventMaker
+secondDayMorning(): Event
+secondDayLunch(): Event
+secondDayAfternoon(): Event
+secondDayAfterSchool(): Event
+thirdDayMorning(): Event
+thirdDayLunch(): Event
+thirdDayAfternoon(): Event
+thirdDayAfterSchool(): Event
+fourthDayMorning(): Event
+fourthDayLunch(): Event
+fourthDayAfternoon(): Event
+fourthDayAfterSchool(): Event
+fifthDayMorning(): Event
+fifthDayLunch(): Event
+fifthDayAfternoon(): Event
+fifthDayAfterSchool(): Event
+sixthDayMorning(): Event
+sixthDayLunch(): Event
+sixthDayAfternoon(): Event
Choice
-choiceName: string
-fate: bool
-affectingStat: bool = false
-affectingCharacter: bool = false
-statToAffect: string
-characterToAffect: Person
-sentences: vector<string>
+Choice()
+Choice(cn:string,f:bool)
+getChoiceName(): string
+getFate(): bool
+setAffectStat(a:bool): void
+setAffectCharacter(a:bool): void
+setStatToAffect(stat:string): void
+setCharacterToAffect(c:Person): void
+addResultSentence(s:string): void
-playChoice(): bool
These 2 classes along with Player and CharacterManager are used throughout the program
StringMagician
+stringToLower(): static string
+bold(): static string
Dialogue
+dialogueContinue(): void
+printFromFile(): string
+highlightWord(): string
+printDialogue(): void
+printDialogue(): void
+getDialogue(): string
+getDialogue(): string

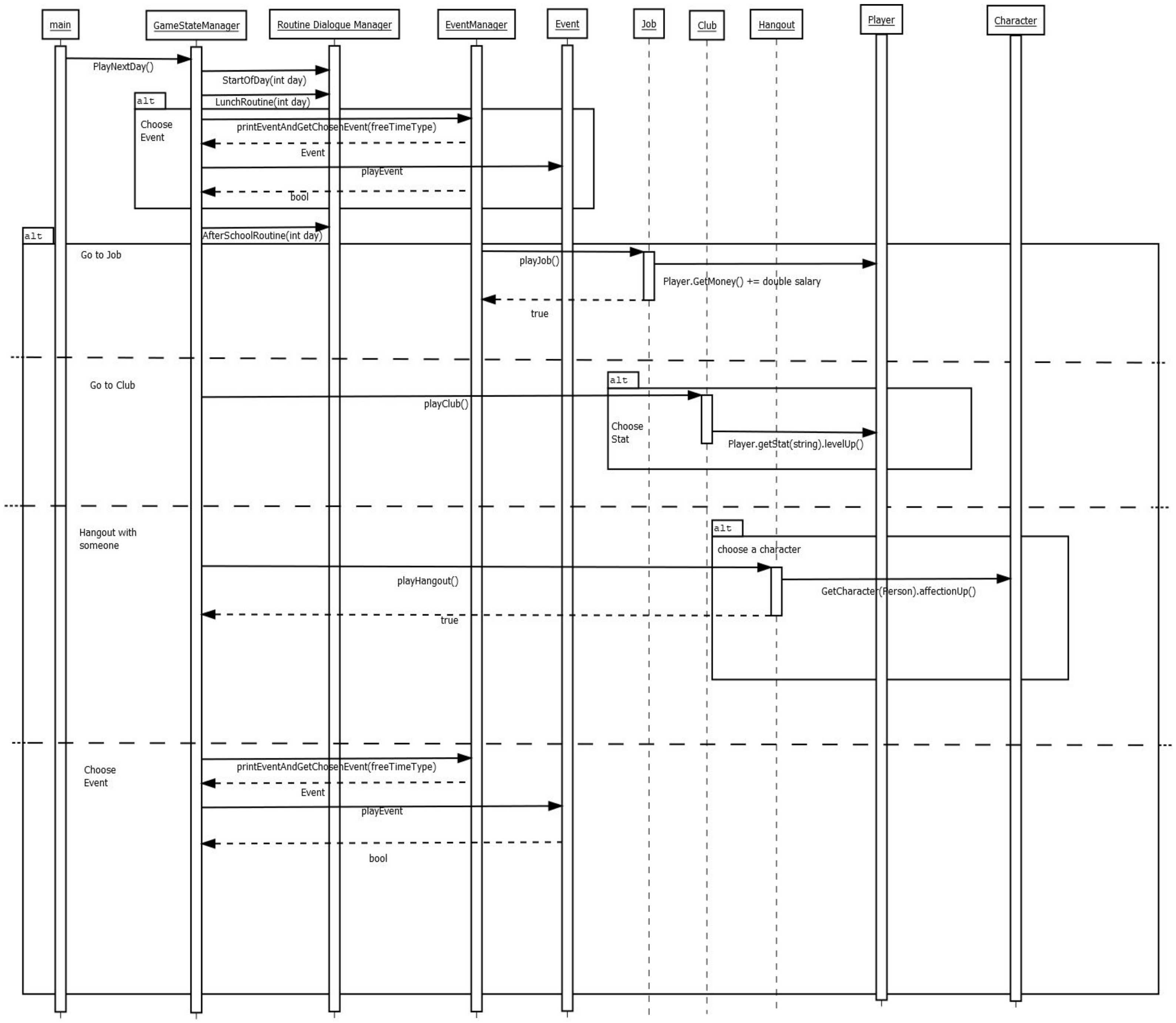
Appendix B

Initialize Game



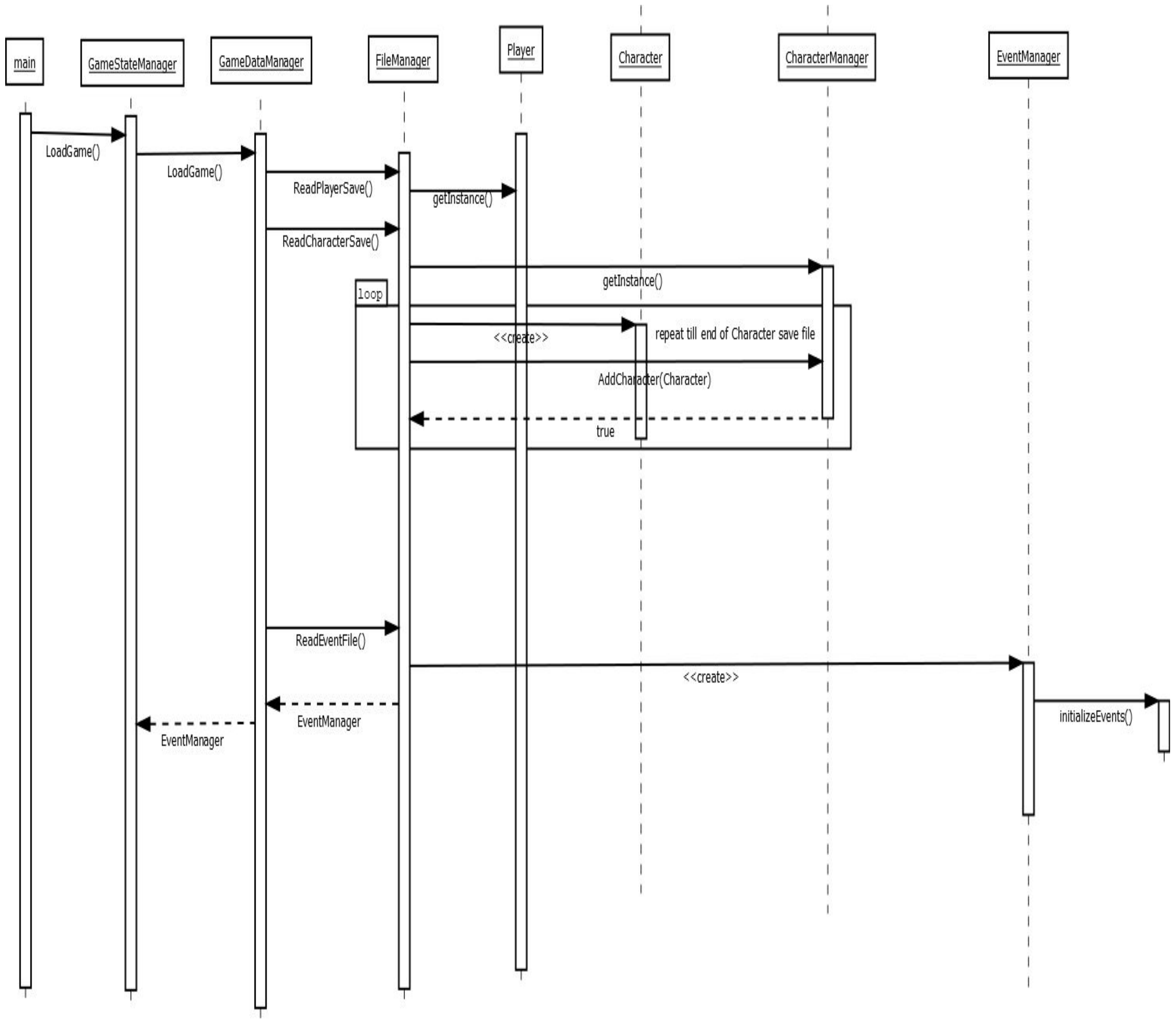
Appendix C

Daily Routine Life



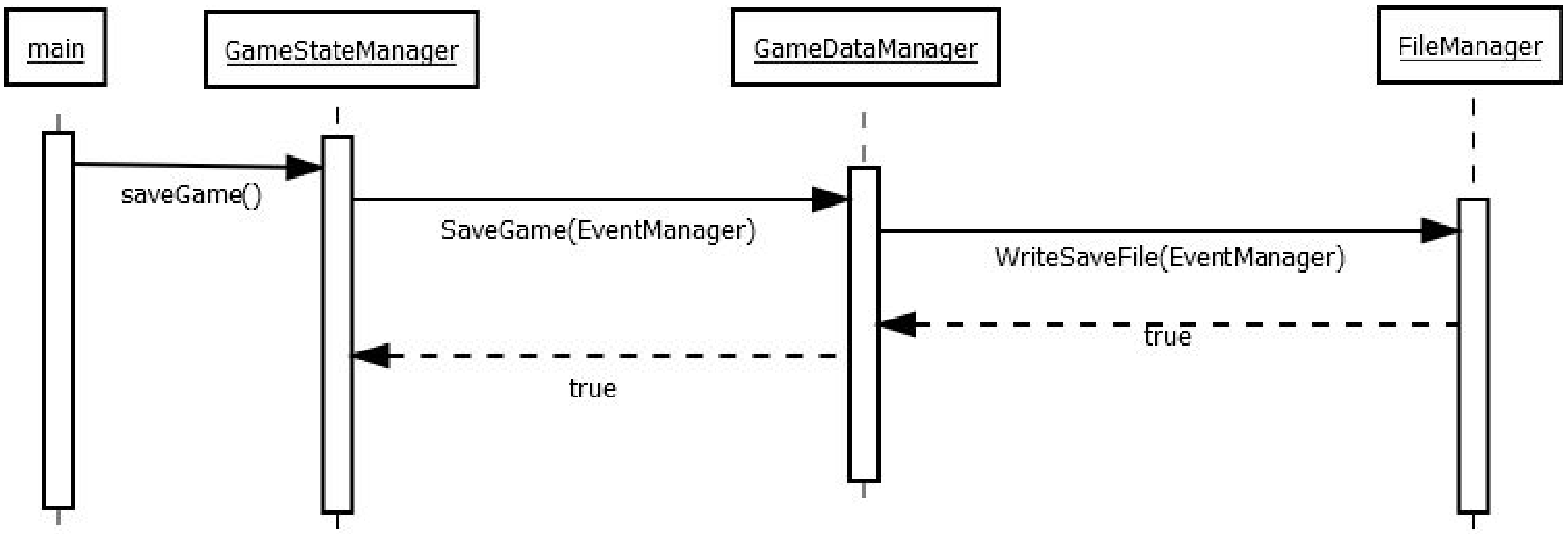
Appendix D

Loading Game



Appendix E

Save Game


main
GameStateManager
GameDataManager
FileManager
saveGame()
SaveGame(EventManager)
WriteSaveFile(EventManager)
true
true