**Software Engineering Project Report**



**A Project Description for   
Development of a new “Travelling” Game**

**Prepared by**

**Kyle Berry, James Kimball, Matthew Runde**

**for**

**Mélange Computing Services**

**September 2014**

**Table of Contents**

[List of Figures 7](#_Toc405672231)

[List of Tables 8](#_Toc405672232)

[I Project Description 9](#_Toc405672233)

[1 Project Overview 9](#_Toc405672234)

[2 The Purpose of the Project 9](#_Toc405672235)

[2a The User Business or Background of the Project Effort 9](#_Toc405672236)

[2b Objectives and Success Criteria of the Project 9](#_Toc405672237)

[3 The Scope of the Work 9](#_Toc405672244)

[3a The Current Situation 9](#_Toc405672245)

[3b The Context of the Work 9](#_Toc405672246)

[3c Work Partitioning 10](#_Toc405672247)

[3d Competing Products 10](#_Toc405672248)

[4 The Scope of the Product 11](#_Toc405672249)

[4a Scenario Diagram(s) 11](#_Toc405672250)

[4b Product Scenario List 11](#_Toc405672251)

[4c Individual Product Scenarios 12](#_Toc405672252)

[5 Stakeholders 15](#_Toc405672253)

[5a The Client 15](#_Toc405672254)

[5b The Customer 15](#_Toc405672255)

[5c Hands-On Users of the Product 15](#_Toc405672256)

[5d Priorities Assigned to Users 19](#_Toc405672257)

[5e User Participation 19](#_Toc405672258)

[5f Maintenance Users and Service Technicians 19](#_Toc405672259)

[5g Other Stakeholders 19](#_Toc405672260)

[6 Mandated Constraints 20](#_Toc405672261)

[6a Solution Constraints 20](#_Toc405672262)

[6b Implementation Environment of the Current System 21](#_Toc405672263)

[6c Partner or Collaborative Applications 21](#_Toc405672264)

[6d Off-the-Shelf Software 21](#_Toc405672265)

[6e Anticipated Workplace Environment 21](#_Toc405672266)

[6f Schedule Constraints 22](#_Toc405672267)

[6g Budget Constraints 22](#_Toc405672268)

[7 Naming Conventions and Definitions 22](#_Toc405672269)

[7a Definitions of Key Terms 22](#_Toc405672270)

[7b UML and Other Notation Used in This Document 23](#_Toc405672271)

[8 Relevant Facts and Assumptions 23](#_Toc405672283)

[8a Assumptions 23](#_Toc405672284)

[II Requirements 24](#_Toc405672285)

[9 Product Use Cases 24](#_Toc405672286)

[9a Use Case Diagrams 24](#_Toc405672287)

[9b Individual Product Use Cases 25](#_Toc405672288)

[10 Functional Requirements 60](#_Toc405672289)

[11 Data Requirements 65](#_Toc405672290)

[12 Performance Requirements 68](#_Toc405672291)

[12a Speed and Latency Requirements 68](#_Toc405672292)

[12b Precision or Accuracy Requirements 68](#_Toc405672293)

[12c Capacity Requirements 68](#_Toc405672294)

[13 Dependability Requirements 68](#_Toc405672295)

[13a Reliability Requirements 68](#_Toc405672296)

[13b Availability Requirements 68](#_Toc405672297)

[13c Safety-Critical Requirements 69](#_Toc405672298)

[14 Maintainability and Supportability Requirements 69](#_Toc405672299)

[14a Maintenance Requirements 69](#_Toc405672300)

[14b Supportability Requirements 69](#_Toc405672301)

[14c Adaptability Requirements 69](#_Toc405672302)

[14d Scalability or Extensibility Requirements 69](#_Toc405672303)

[14e Longevity Requirements 69](#_Toc405672304)

[15 Security Requirements 70](#_Toc405672305)

[15a Access Requirements 70](#_Toc405672306)

[15b Integrity Requirements 70](#_Toc405672307)

[15c Privacy Requirements 70](#_Toc405672308)

[15d Audit Requirements 70](#_Toc405672309)

[15e Immunity Requirements 70](#_Toc405672310)

[16 Usability and Humanity Requirements 70](#_Toc405672311)

[16a Ease of Use Requirements 70](#_Toc405672312)

[16b Personalization and Internationalization Requirements 71](#_Toc405672313)

[16c Learning Requirements 71](#_Toc405672314)

[16d Understandability and Politeness Requirements 71](#_Toc405672315)

[16e Accessibility Requirements 71](#_Toc405672316)

[16f User Documentation Requirements 71](#_Toc405672317)

[16g Training Requirements 72](#_Toc405672318)

[17 Look and Feel Requirements 72](#_Toc405672319)

[17a Appearance Requirements 72](#_Toc405672320)

[17b Style Requirements 72](#_Toc405672321)

[18 Operational and Environmental Requirements 72](#_Toc405672322)

[18a Expected Physical Environment 72](#_Toc405672323)

[18b Requirements for Interfacing with Adjacent Systems 73](#_Toc405672324)

[18c Productization Requirements 73](#_Toc405672325)

[18d Release Requirements 73](#_Toc405672326)

[19 Cultural and Political Requirements 73](#_Toc405672327)

[19a Cultural Requirements 73](#_Toc405672328)

[19b Political Requirements 74](#_Toc405672329)

[20 Legal Requirements 74](#_Toc405672330)

[20a Compliance Requirements 74](#_Toc405672331)

[20b Standards Requirements 74](#_Toc405672332)

[III Design 74](#_Toc405672333)

[21 System Design 74](#_Toc405672334)

[21a Design goals 74](#_Toc405672335)

[22 Current Software Architecture 76](#_Toc405672336)

[23 Proposed Software Architecture 76](#_Toc405672337)

[23a Overview 76](#_Toc405672338)

[23b Class Diagrams 77](#_Toc405672339)

[23c Dynamic Model 79](#_Toc405672340)

[23d Subsystem Decomposition 88](#_Toc405672341)

[23e Hardware / software mapping 89](#_Toc405672342)

[23f Data Dictionary 89](#_Toc405672343)

[23g Persistent Data management 91](#_Toc405672344)

[Long Term Data Storage: Stores the state of a game session, including all of the other persistent data stored by any class. Its data is only accessed when loading a game, and is only changed when saving the game. In other words, this is a saved game file. 92](#_Toc405672345)

[23h Access control and security 93](#_Toc405672346)

[23i Global software control 93](#_Toc405672347)

[23j Boundary conditions 93](#_Toc405672348)

[24 Subsystem services 94](#_Toc405672349)

[25 User Interface 94](#_Toc405672350)

[26 Object Design 95](#_Toc405672351)

[26a Object Design trade-offs 95](#_Toc405672352)

[26b Interface Documentation guidelines 95](#_Toc405672353)

[26c Packages 96](#_Toc405672354)

[26d Class Interfaces 97](#_Toc405672355)

[IV Test Plans 108](#_Toc405672356)

[27 Features to be tested / not to be tested 108](#_Toc405672357)

[28 Pass/Fail Criteria 108](#_Toc405672358)

[29 Approach 109](#_Toc405672359)

[30 Suspension and resumption 109](#_Toc405672360)

[31 Testing materials ( hardware / software requirements ) 109](#_Toc405672361)

[32 Test cases 110](#_Toc405672362)

[33 Testing schedule 120](#_Toc405672363)

[V Project Issues 121](#_Toc405672364)

[34 Open Issues 121](#_Toc405672365)

[35 Off-the-Shelf Solutions 121](#_Toc405672366)

[35a Ready-Made Products 121](#_Toc405672367)

[35b Reusable Components 121](#_Toc405672368)

[35c Products That Can Be Copied 121](#_Toc405672369)

[36 New Problems 121](#_Toc405672370)

[36a Effects on the Current Environment 121](#_Toc405672371)

[36b Effects on the Installed Systems 122](#_Toc405672372)

[36c Potential User Problems 122](#_Toc405672373)

[36d Limitations in the Anticipated Implementation Environment That May Inhibit the New Product 122](#_Toc405672374)

[36e Follow-Up Problems 122](#_Toc405672375)

[37 Tasks 122](#_Toc405672376)

[37a Project Planning 122](#_Toc405672377)

[37b Planning of the Development Phases 123](#_Toc405672378)

[38 Migration to the New Product 123](#_Toc405672379)

[38a Requirements for Migration to the New Product 123](#_Toc405672380)

[38b Data That Has to Be Modified or Translated for the New System 123](#_Toc405672381)

[39 Risks 123](#_Toc405672382)

[40 Costs 124](#_Toc405672383)

[41 Waiting Room 124](#_Toc405672384)

[42 Ideas for Solutions 125](#_Toc405672385)

[43 Project Retrospective 125](#_Toc405672386)

[VI Glossary 125](#_Toc405672387)

[VII References / Bibliography 125](#_Toc405672388)

[VIII Index 126](#_Toc405672389)

### ****List of Figures****

### ****List of Tables****

# Project Description

## Project Overview

The purpose of this project is to create a software gaming package for a time-traveling strategy game that revolves around a single individual manipulating events in the past to change the present and future, taking advantage of time itself to greatly change the world despite having few powers beyond temporal movement. The goal of the game is variable, as the game can use different parameter list sets to change the nature of the game. For example, one sample parameter lists set might detail the rules for a world where the goal is successfully engineer the creation of an empire that can fend off an alien invasion that occurs in the year 2000.

## The Purpose of the Project

### The User Business or Background of the Project Effort

As a software development group creating a game, our business involves creating a software package that will sell well, establish a well-liked brand name, and create additional opportunities to for profit. Most users of this software are expected to use the game for recreational purposes, while others may additionally take advantage of the software to create third party additions that enhance the experience for themselves and others (which in turn will empower the name of our brand).

### Objectives and Success Criteria of the Project

The goal of the project is to create a game that involves time travel. The game must be unique\* and fun\*\* to play for both short and long periods of time. Furthermore, the game must create a platform from which we can market additional products while also being fun\*\* for the user that does not wish to buy any additional products.

* “Unique” refers to the program as a whole – as long as there is not another game that is using time travel the same way we are the game will be considered unique.
* “Fun” will be defined as attaining reviews from focus groups asked to compare this game with other games and recreational activities via survey that suggest this game is about as/more enjoyable than our major competitors.

Measurement

Any reasonable goal must be measurable. This is necessary if you are ever to test whether you have succeeded with the project. The measurement must quantify the *advantage* gained by the business through doing the project. If the project is worthwhile, there must be some solid business reason for doing it. For example, if the goal of the project is

We want to give immediate and complete response to customers who order our goods over the telephone.

you have to ask what advantage that goal brings to the organization. If immediate response will result in more satisfied customers, then the measurement must quantify that satisfaction. For example, you could measure the increase in repeat business (on the basis that a happy customer comes back for more), the increase in customer approval ratings from surveys, the increase in revenue from returning customers, and so on.

It is crucial to the rest of the development effort that the goal is firmly established, is reasonable, and is measured. It is usually the latter that makes the former possible.

## The Scope of the Work

### The Current Situation

The software package is not intended to replace or change any existing manual or automated processes.

### The Context of the Work

The gaming application will be a time travelling game. The player will be able to navigate to different times in history and change certain aspects in each time zone. Aspects that are changed will change events and image files according to what the player has done.

### Work Partitioning

* + - 1. Generating New Content for the Game
      2. Distributing/Selling New Content

### Competing Products

Specific games that have some similarity to this one include the Civilization Series (overseeing a nation/worlds technological development) and any game that revolves around time travel. However, we do not know of any game that combines Civilization-esque world building with time travel. The experience could be somewhat approximated by playing a game of civilization with several save files, but the result would be very lacking compared to what this game will offer.

However! Our package is competing for the recreational time and funds of our customers. Thus, any and every form of entertainment the customer could utilize to fulfill their recreational needs could be viewed as our competition. Computer games do have an intrinsic advantage over movie theaters and other activities that cannot be done from the user’s home, so we will largely focus on other electronic games when speaking of the competition.

Furthermore, although being used for educational purposes is not a primary goal of our product we should keep in mind other software programs that offer educational content.

## The Scope of the Product

### Scenario Diagram(s)

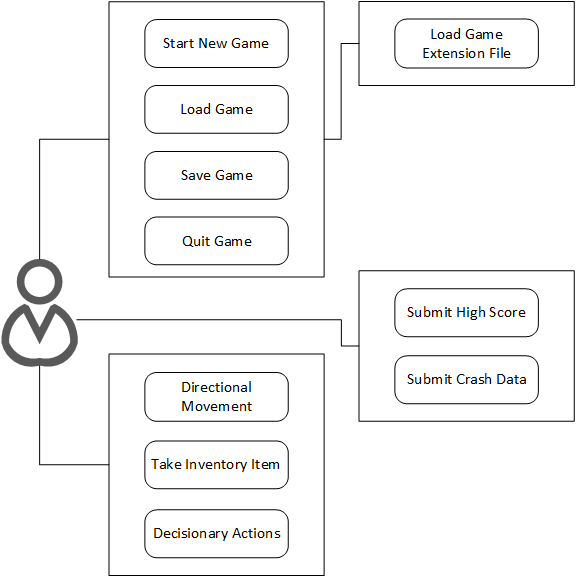


Figure 1: User interaction diagram

### Product Scenario List

* User Scenario – Start New Game
* User Scenario – Load Game
* User Scenario – Save Game
* User Scenario – Quit Game
* User Scenario – Directional Movement
* User Scenario – Take Inventory Item
* User Scenario – Decisionary Actions

### Individual Product Scenarios

**Start New Game**

|  |  |
| --- | --- |
| Scenario name | Start New Game |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew has created a user account and downloaded the game from the website. |
| Flow of Events | 1) Andrew clicks on the gaming application for the first time.  2) Temporal Architect presents a start game page.  3) Andrew clicks on “Start New Game”.  4) Temporal Architect starts a new game. |

**Load Game**

|  |  |
| --- | --- |
| Scenario name | Load Game |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew has played the game previously and saved his gameplay. |
| Flow of Events | 1) Andrew clicks on the gaming application.  2) Temporal Architect presents a start game page.  3) Andrew clicks on “Load Game”  4) Temporal Architect switches to a page with a list of saved games.  5) Andrew clicks the game he wishes to continue to play.  5) Temporal Architect loads the game. |

**Save Game**

|  |  |
| --- | --- |
| Scenario name | Save Game |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew has played the game and is ready to save his gameplay. |
| Flow of Events | 1) Andrew clicks the “esc” button.  2) Temporal Architect pulls up a pause menu.  3) Andrew selects “save game” from the menu.  4) Temporal Architect saves the game and lets Andrew know that the game has been saved. |

**Quit Game**

|  |  |
| --- | --- |
| Scenario name | Quit Game |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew has finished playing the game and is ready to quit. |
| Flow of Events | 1) Andrew clicks the “esc” button.  2) Temporal Architect pulls up a pause menu.  3) Andrew selects “Quit Game” from the menu.  4) Temporal Architect closes. |

**Submit High Score**

|  |  |
| --- | --- |
| Scenario name | Submit High Score |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew has died and is no longer able to continue. |
| Flow of Events | 1) Temporal Architect brings up a screen asking “start over”, “submit high score”, and “save and quit”.  2) Andrew selects “submit high score”  3) Temporal Architect submits the score to the database. |

**Submit Crash Data**

|  |  |
| --- | --- |
| Scenario name | Submit Crash Data |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew was playing when the game crashed. |
| Flow of Events | 1) Temporal Architect will ask “Would you like to submit the crash data to the company for further review?”  2) Andrew clicks “yes”. |

**Directional Movement**

|  |  |
| --- | --- |
| Scenario name | Directional Movement |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew is playing the game |
| Flow of Events | 1) Andrew clicks to move north.  2) Andrew clicks to move south.  3) Andrew clicks to move west.  4) Andrew clicks to move east. |

**Take and Inventory Item**

|  |  |
| --- | --- |
| Scenario name | Take and Inventory Item |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew finds an item and wants to add it to his inventory. |
| Flow of Events | 1) Andrew clicks to pick up the item.  2) Temporal Architect saves the item in the inventory list. |

**Decisionary Actions**

|  |  |
| --- | --- |
| Scenario name | Decisionary Actions |
| Participating Actor | Andrew: Player |
| Precondition(s) | Andrew is interacting with an NPC in the game. |
| Flow of Events | 1) The NPC asks Andrew a question.  2) Andrew selects an answer from the list of answers. |

## Stakeholders

### The Client

The client, Mélange Computing Services, serves as the main stakeholder and provides the bulk of the funding and the application shall conform to their main requirements. As a major computing company that only recently entered the gaming industry this is an opportunity to form a lasting relationship with the company as one of their intitial titles.

### The Customer

Parents/Teachers of Children, Teens, and Young Adults of both genders, including those not currently interested in gaming.

Older adults that are already interested in gaming (we will not be trying to lure more of them into the gaming community).

### Hands-On Users of the Product

AIIG = “Already Interested in Gaming”

**Children**:

Role: Play the game. Children are not expected to look online for outside help nor are they expected to be patient when dealing with confusing or unclear instructions.

Subject Matter Experience: Novice

Technological Experience: Novice

Other Characteristics: Appealing to the parents and teachers of children by offering educational content (such as by promoting critical thinking) could be of great use – but only if our program can run on the old computers schools have and is not prohibitively expensive.

Children are unlikely to buy a game with their own money, but might be able to talk their parents into doing so.

**Male Teens** (AIIG):

Role: Play the game. Teens that are interested in gaming might look online for outside help and can be expected to be somewhat patient and willing to learn when exposed to initially confusing or unclear instructions. Such teens might also develop new content for the game for both recreational and commercial use.

Subject Matter Experience: Journeyman -Teens will likely have some prior knowledge of the kinds of cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Master – Teens have a great deal of free time to dedicate to mastering recreational games.

Other Characteristics: More research is needed to determine whether this type of game appeals more to one gender than another.

**Female Teens** (AIIG):

Role: Teens that are interested in gaming might look online for outside help and can be expected to be somewhat patient and willing to learn when exposed to initially confusing or unclear instructions. Such teens might also develop new content for the game for both recreational and commercial use.

Subject Matter Experience: Journeyman -Teens will likely have some prior knowledge of the kinds of cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Master – Teens have a great deal of free time to dedicate to mastering recreational games.

Other Characteristics: More research is needed to determine whether this type of game appeals more to one gender than another.

**Teens** (!AIIG):

Role: Play the game. Teens that are not interested in gaming are not expected to look online for outside help nor are they expected to be patient when dealing with confusing or unclear instructions.

Subject Matter Experience: Journeyman -Teens will likely have some prior knowledge of the kinds of cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Novice - Although teens have a great deal of free time, those not interested in gaming probably didn’t spend it mastering games.

Other Characteristics: We are assuming that the difference between being interested in games VS not being interested is much greater than any gender preference. However, this may be an incorrect assumption in which case this group should be split into two gender-specific groups.

Teens that are not interested in gaming may become interested in gaming as a result of playing this game.

**Male Young Adults** (AIIG):

Role: Young Adults that are interested in gaming might look online for outside help and can be expected to be patient and willing to learn when exposed to initially confusing or unclear instructions. They have likely played many games in the past and thus are more likely than teens to notice similarities to other games and may be more critical of the content of this game. Such users might also develop new content for the game for both recreational and commercial use.

Subject Matter Experience: Journeyman - Young adults that are interested in gaming have less free time to spend mastering a game than teens but have more life experiences (including gaming-specific ones) to draw from when it comes to the many cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Journeyman - Young adults that are interested in gaming have less free time to spend mastering a game than teens but have more life experiences (including gaming-specific ones) to draw from.

**Female Young Adults** (AIIG):

Role: Young Adults that are interested in gaming might look online for outside help and can be expected to be patient and willing to learn when exposed to initially confusing or unclear instructions. They have likely played many games in the past and thus are more likely than teens to notice similarities to other games and may be more critical of the content of this game. Such users might also develop new content for the game for both recreational and commercial use

Subject Matter Experience: Journeyman - Young adults that are interested in gaming have less free time to spend mastering a game than teens but have more life experiences (including gaming-specific ones) to draw from when it comes to the many cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Journeyman - Young adults that are interested in gaming have less free time to spend mastering a game than teens but have more life experiences (including gaming-specific ones) to draw from.

Other Characteristics: More research is needed to determine whether this type of game appeals more to one gender than another

**Young Adults** (!AIIG):

Role: Young Adults that are interested in gaming might look online for outside help and can be expected to be somewhat patient and willing to learn when exposed to initially confusing or unclear instructions. Such users might also develop new content for the game for both recreational and commercial use.

Subject Matter Experience: Journeyman - Young adults that are interested in gaming have less free time to spend mastering a game than teens but have more life experiences to draw from when it comes to the many cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Novice - Young adults that are not interested in games likely lack both the past experience with gaming and the time needed to master the related technologies now.

Other Characteristics: We are assuming that the difference between being interested in games VS not being interested is much greater than any gender preference. However, this may be an incorrect assumption in which case this group should be split into two gender-specific groups.

Adults that are not interested in gaming may become interested in gaming as a result of playing this game.

**Older Adults** (AIIG):

Role: Adults that are interested in gaming might look online for outside help and can be expected to be patient and willing to learn when exposed to initially confusing or unclear instructions. They have likely played many games in the past and thus are more likely than teens to notice similarities to other games and may be more critical of the content of this game. Such users might also develop new content for the game for both recreational and commercial use

Subject Matter Experience: Master - Adults that are interested in gaming have less free time to spend mastering a game than teens but have far more life experiences to draw from (including gaming experiences) when it comes to the many cause-and-effect relationships they are manipulating via time travel.

Technological Experience: Journeyman - Adults that are interested in gaming have less free time to spend mastering a game than teens but have more life experiences (including gaming-specific ones) to draw from.

Other Characteristics: This group may need to be further divided based on gender.

### Priorities Assigned to Users

**KEY USERS**: Young Adults, Teens, and Adults that are already interested in gaming.

**SECONDARY USERS**: Children and their parents. Young Adults and Teens that are not already interested in gaming.

**UNIMPORTANT USERS**: Adults that have no interest / negative interest in gaming (unless parents of children).

### User Participation

The software package will rely heavily on user input. Options for user input will be made available on the company website (through the use of forums and rating options). This user input, alongside the crash data, will allow the developers to improve and fix user-found bugs in the software in future releases.

Users will also have the opportunity to beta test the software packages on an invitation only basis. Users which accept the invitation will have access to internal blogs, forums, and bug tracking lists which shall be read and visited at a greater rate by the developing teams.

### Maintenance Users and Service Technicians

Maintenance users will be any user who wants to create their own .gdf file for themselves or others to play with. Such users will need to know how to format such files, use a text editor, use their created files with the game, and possibly how to share these files with others.

Service technicians will be used to make sure the company website and company server are running according to specs. They will need to know how to maintain websites/servers in addition to being familiar enough with the game itself and the users of the game to understand what parts of the websites/servers are the most important. A website that looks nice but is a pain for customers to use is a waste of money.

### Other Stakeholders

**Stakeholder**: Software Developers

**Knowledge Needed By Project**: How much time do we expect each aspect of the program will take to implement? What code segments are more vital to the overall project than others? What libraries if any will be using, what will be the standard coding/commenting style for this project, etc.

**Degree of Involvement**: 5/5

**Degree of Influence**: 4/5

**Additional Comments**: n/a

**Stakeholder**: Educational Institutions

**Knowledge Needed By Project**: What do educational institutions look for in a game? How do they choose one game over another? Who makes these decisions?

**Degree of Involvement**: 1/5

**Degree of Influence**: 2/5

**Additional Comments**: n/a

**Stakeholder**: Game Distributors

**Knowledge Needed By Project**: How does game distribution work? How do we convince them to distribute our game? Are some methods of distribution traditionally more successful than others? To what degree type of game / intended consumer / etc effect optimal distribution method?

**Degree of Involvement**: 2/5

**Degree of Influence**: 3/5

**Additional Comments**: n/a

**Stakeholder**: Public Watchdog Groups

**Knowledge Needed By Project**: What things do these groups consider unacceptable that the general public can be pushed to care about as well?

**Degree of Involvement**: 1/5 - They will not be involved at all until after the game is developed.

**Degree of Influence**: 1/5 - We should avoid having anything in the game that is especially inciting.

**Additional Comments**: n/a

## Mandated Constraints

### Solution Constraints

The software package has three primary technological constraints. One constraint will be the software package for the client. This package will contain an easily downloadable copy of the official game and .gdf file. Another constraint will be to create a company web page that will show information about the game, information about the company, and statistics on the top players. The final constraint will be creating a company server that will be used to collect all client statistics, crash data, and send out updates to the game file as they come about. The server will also communicate with the company website to update player statistics.

### Implementation Environment of the Current System

The software package is intended to be installed onto a personal computer’s (PC) storage media. The installation location does not necessarily need to reside on the bootable storage media to run, however, an operating system does need to exist (as specified) to run and/or operate.

The software package will also require a standard media output device (such as a standard VGA monitor).

### Partner or Collaborative Applications

The user may choose to have the software package will collaborate with the software server, which also hosts a public website. The user may play the game offline, however. If collaboration is preferred then the software package will communicate with the server upon application exit and send information about user statistics, which will be displayed on the webpage for other users to view.

If the user chooses collaborations they may also access our company’s virtual store for the game. However, the access must be unlocked from our website (possibly while purchasing the game) and is not immediately made available due to potential misuse by children.

### Off-the-Shelf Software

The software package requires either Windows Vista, 7, Windows 8.x (or above), Linux, and/or RedHat be installed on the computer.

Creating new .gdf files for the game requires the user have a working text editor (all of the above operating systems come with text editors, but some users might remove them on accident).

### Anticipated Workplace Environment

The software package will typically be used in a home environment for personal entertainment. The household might have internet capabilities to allow the software package to connect to the software server and send user statistics. If the client is not connected to the internet, space will be allocated to save game state and client statistics locally.

### Schedule Constraints

The software package should meet the scheduled internal development release dates as specified in the *Agile* development requirements in order to meet the investors and publishers release date requirements.

### Budget Constraints

The development cycle of the software package should be kept to a minimum while maintaining the *Agile* development cycle. The budget for the development cycle (including sound and artwork) shall not exceed 600,000 US dollars.

## Naming Conventions and Definitions

### Definitions of Key Terms

AIIG = “Already Interested in Gaming”

Game Client = The game.

Flagged = A variable noting that something is true/false has been set to True (or otherwise that the game somehow knows that whatever has been ‘Flagged’ is True, depending on exact implementation).

Avatar = The individual who the User controls in the game world.

User Actions = Actions that the User’s avatar has taken over the course of the game. A list of these are stored for each in-game year in the Short Term History architecture.

Set Maxed Date = The farthest date in the future the User can reach. For example, perhaps they can only go as far as year 5000.

Changed Future = A future (in game) that might be different do to something having been changed in an earlier year.

Unreachable = A year that cannot be reached due to a limitation in the Parameter Files besides being beyond the Set Maxed Date / Before Year Zero. For example, perhaps it cannot be reached because the planet is currently destroyed the previous year.

Square = A square subsection of the world map. How many squares make up a world is determined via the parameter files.

Parameter Files = The files that detail victory conditions, scoring, what objects exist, what actions are available, the results of any given action, what the starting conditions are, etc. They are effectively the rules for any given game world.

Object = An item of some kind, such as wood, a chest, a gun, a helicopter, etc. What Objects can exist in any given game world and their properties are determined by the Parameter Files. Thus, an object can be nearly anything. Even magical spellbooks and the like are technically possible if described in the Parameter Files.

Geographical Features = Examples include rivers, forests, volcanoes, etc. What features can exist are defined in the Parameter Files.

NPC = Non Player Character. Communities are treated as a single NPC, as are individuals although they have different capabilities.

!<Any other term> = Not <Whatever the other term is describing>. For example: !AIIG would mean “Not Already Interested in Gaming

Fun = Attaining reviews from focus groups asked to compare this game with other games and recreational activities via survey that suggest this game is about as/more enjoyable than our major competitors.

Enjoyable = A property of being pleasurable to interact with.

Unique = Refers to the program as a whole – as long as there is not another game that is using time travel the same way we are the game will be considered unique.

Agile Development = A group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

Friendliest Ratings = There are many different rating systems used by the gaming industry that determine how old a child should be before it is okay to expose them to this game’s content. The “Friendliest’ rating to the one that corresponds to being safe to use by any subset of the population.

### UML and Other Notation Used in This Document

This document generally follows the Version 2.0 OMG UML standard, as described by Fowler in [4]. Any exceptions are noted where used.

Data Dictionary for Any Included Models

Content

Dictionary definitions of all information flows and stores used in models. Particular consideration should be given to defining the data attributes of all flows shown the context models (see sections 7 and 8).

This section should also contain any technical specifications for interfaces shown on the context models.

Motivation

The context diagram provides an accurate definition of the scope of the work being studied or the scope of the product to be built. This definition can be completely accurate only if the information flows bordering the scope have their attributes defined.

Examples

Road de-icing schedule = issue number + {road section identifier + treatment start time + critical start time + truck identifier} + depot identifier

As you progress through the requirements specification, define each of the elementary terms in detail.

Considerations

The dictionary provides a link between the requirements analysts and the implementers. The implementers add implementation details to the terms in the dictionary, defining how the data will be implemented. Also, implementers add terms that are present because of the chosen technology and that are independent of the business requirements.

## Relevant Facts and Assumptions

### Assumptions

The software package will be more successful if it achieves the friendliest ratings (Such as “E for everyone”) from groups such as ESRB.

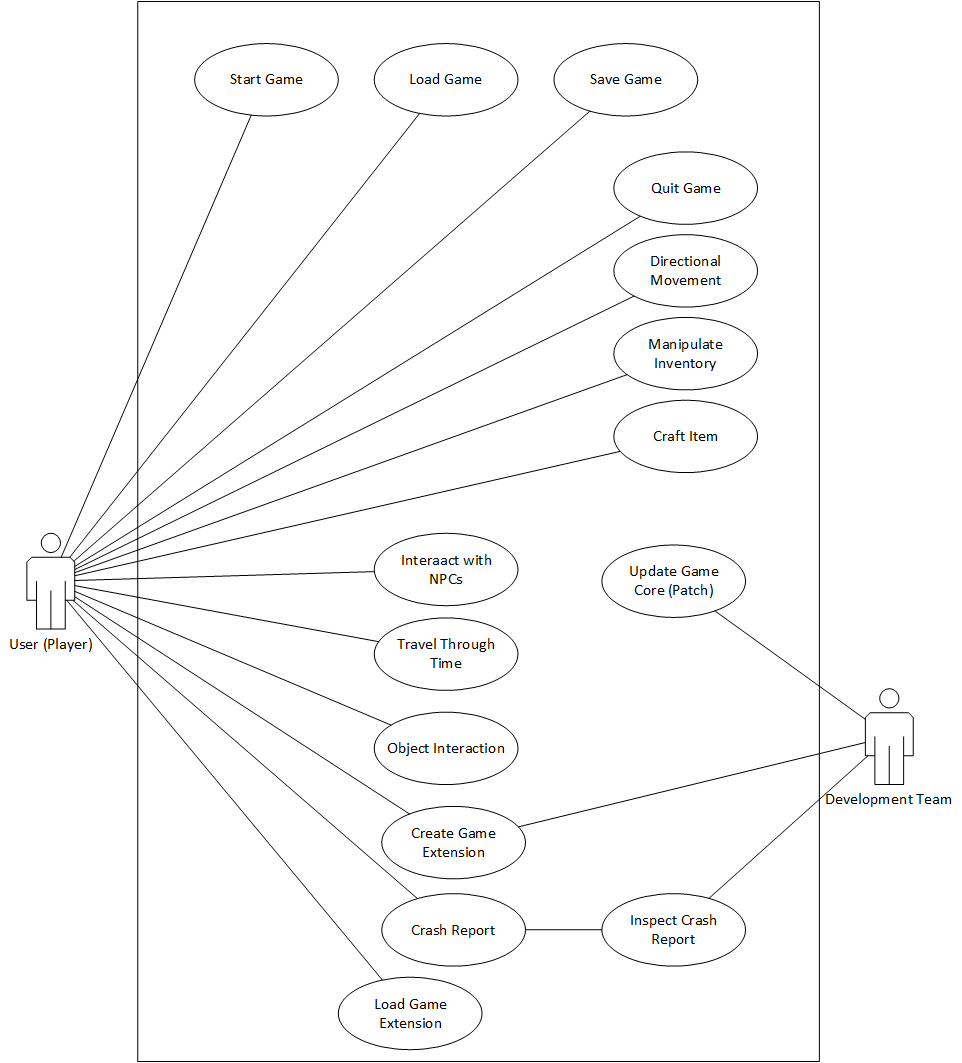
The software package should be easy to develop additional content for.

The software package should not require additional content beyond that supplied during the initial purchase to meet its design goals and constraints. This assumption does not include design goals and constraints that specifically involve the creation and purchase of additional content.

# Requirements

## Product Use Cases

### Use Case Diagrams

****

### Individual Product Use Cases

The product use cases are separated into two groups: The first group deals with working with the product in ways besides playing the game (update, modifications, crash report data, etc.), while the second group of scenarios deals with the user performing actions while playing the game.

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – UnrestrictedOverlandMovement |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar travel North, South, East, or West.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then one of the following:‘1’ for North, ‘2’ for South, ‘3’ for East, or ‘4’ for West) or presses a previously set hotkey.  3. The Game Client checks the parameters for this world to see if there are any special movement restrictions. It does not find any.  4. The Game Client updates the world map to reflect the Avatar’s new location and updates the User’s available options accordingly. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User desires to have the Avatar move north, south, east, or west to an adjacent square.  > The parameters used in this particular game session does not possess any restrictions on movement. |
| Exit Condition | > The User’s Avatar is now in the square one step in the chosen direction from their previous square and can perform any other actions that are available at this new current location at their current point in history. |
| Quality  Requirements | The world should wrap around – I.E., it should be impossible to have a situation where it is impossible to move in one of the cardinal directions.. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – RestrictedOverlandMovement |
| Participating Actors | User |
| Flow  of Events | 1. User decides to have their Avatar travel North, South, East, or West.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then one of the following:‘1’ for North, ‘2’ for South, ‘3’ for East, or ‘4’ for West) or presses a previously set hotkey.  3. The Game Client checks the parameters for this world to see if there are any special movement restrictions. It finds one.  4. The Game Client checks to see if the desired movement would violate one or more restrictions (perhaps some squares are off limits or a player only has a certain number of moves per year.). It confirms the movement would violate a restriction.  5. The Game Client displays an error message stating that the desired movement is not possible. The exact contents of the error message are drawn from the “reason” variable of that restriction’s class. If the variable is null, the error message simply states that the current year is unreachable. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User desires to have the Avatar move north, south, east, or west to an adjacent square.  > The parameters used in this particular game session list include restriction(s) that impedes movement.  >The above restrictions happen to make the User’s desired movement impossible. |
| Exit Condition | > The User’s Avatar is remains at its current location and can perform any other actions that are available at its current location at its current point in history. |
| Quality  Requirements |  |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Restricted But Allowed OverlandMovement |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar travel North, South, East, or West.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then one of the following:‘1’ for North, ‘2’ for South, ‘3’ for East, or ‘4’ for West) or presses a previously set hotkey.  3. The Game Client checks the parameters for this world to see if there are any special movement restrictions. It finds one.  4. The Game Client checks to see if the desired movement would violate one or more restrictions (perhaps some squares are off limits or a player only has a certain number of moves per year.). It confirms the movement does not violate a restriction.  5. The Game Client updates the world map to reflect the Avatar’s new location and updates the User’s available options accordingly. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User desires to have the Avatar move north, south, east, or west to an adjacent square.  > The parameters used in this particular game session list include restriction(s) that impedes movement.  > The desired movement is still permissible under the restriction(s). |
| Exit Condition | > The User’s Avatar is now in the square one step in the chosen direction from their previous square and can perform any other actions that are available at this new current location at their current point in history. |
| Quality  Requirements | The world should wrap around – I.E., it should be impossible to have a situation where it is impossible to move in one of the cardinal directions.. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Traveling From a Year To An Earlier Year That Is Not Before Year 0. |
| Participating Actors | User |
| Flow of Events | 1. User decides to have Avatar travel to the past.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey.  3. The Game Client displays a prompt requesting a date.  4. User types in a date that is earlier than the current in-game date.  5. The Game Client, recognizing that the entered date is earlier than the current date but greater than or equal to 0, checks its saved data for new data and re-loads the world in accordance to the data. The only exception is the User’s inventory, which remains the same. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The year the User wishes to go to before the date their Avatar is currently at is >= 0. |
| Exit Condition | The User’s Avatar is now at the specified past date and can perform any other actions that are available at their current location at that point in history. |
| Quality  Requirements | This process should not take longer than 10 seconds. |

|  |  |
| --- | --- |
| Use Case Name | Playing Game – Attempting To Travel To A Year Before Year 0. |
| Participating Actors | User, |
| Flow of Events | 1. User decides to have Avatar travel to the past.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey.  3. The Game Client displays a prompt requesting a date.  4. User types in a date that is earlier than the current in-game date.  5. The Game Client, recognizing that the entered date is less than the current data but is also less than 0, displays an error message stating that the desired year cannot be reached under any circumstances. |
| Entry Condition | The User has started the game and has either loaded or initialized a world to play in. |
| Exit Condition | The User’s Avatar remains at their current date and can perform any other actions that are available at their current location at that point in history. |
| Quality  Requirements |  |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Traveling To an Unchanged Future That Is Not Flagged “Unreachable” and is Earlier Than “Set Maxed Date” |
| Participating Actors | User, |
| Flow of Events | 1. User decides to have their Avatar travel to the future.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey.  3. The Game Client displays a prompt requesting a date.  4. User types in a date that is later than the current in-game date.  5. The Game Client, recognizing that the entered date is later than the current date but less than the Set Maxed Date, checks to see if any of the years between the current date and the entered date have been flagged with ‘changed’. The check reveals no years have been so flagged.  6. The Game Client re-loads the world in accordance to the saved data for the entered year. The only exception is the User’s inventory, which remains the same. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User has not done anything that has caused the game to flag a year between their current year and the one they wish to go to (including the one they wish to go to) with ‘changed’. |
| Exit Condition | The User’s Avatar is now at the specified future date and can perform any other actions that are available at their current location at that point in history. |
| Quality  Requirements | This process should not take longer than 10 seconds. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Traveling To an Changed Future that is not Flagged “Unreachable” and is Earlier Then “Set Maxed Date” |
| Participating Actors | User, |
| Flow of Events | 1. User decides to have their Avatar travel to the future.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey.  3. The Game Client displays a prompt requesting a date.  4. User types in a date that is later than the current in-game date.  5. The Game Client, recognizing that the entered date is later than the current date but less than the Set Maxed Date, checks to see if any of the years between the current date and the entered date have been flagged with ‘changed’. The check reveals that one or more years have been flagged.  6. Beginning with the first flagged year and ending with the entered date, the game performs the following actions for each year in order from ‘i’ to ‘vi’:  --------i) Checks to see if flagged year is also flagged ‘Unreachable’. As per the entry conditions, all such checks return ‘false’ and no action is taken.  --------ii) Re-calculates the state of the world for the flagged year using the information of the previous year.  --------iii) Attempts to apply any stored User Actions for the flagged year to the state of the world in the flagged year. If the preconditions of the stored User Action are not meant (such as redirecting a river that no longer exists), then it is skipped.  -------iv) The year’s ‘changed’ flag is set to ‘false’.  -------v) The next year’s ‘changed’ flag is set to ‘true’.  -------vi) The next year is checked to see if it has the ‘Unreachable’  7. The Game Client re-loads the world in accordance to newly modified saved data for the entered year. The only exception is the User’s inventory, which remains the same. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User has performed one or more actions that has caused the game to flag a year between their current year and the one they wish to go to (including the one they wish to go to) with ‘changed’.  > The year the User wishes to go to is not flagged ‘unreachable’ and is both >= 0 and < Maxed Set Date. |
| Exit Condition | > The User’s Avatar is now at the specified future date and can perform any other actions that are available at their current location at that point in history.’  > All years between the current date and destination year have been updated and are no longer flagged as changed. |
| Quality  Requirements | This process should not take longer than 10 seconds. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Attempting To Travel To A Year That is Later Than “Set Maxed Date” |
| Participating Actors | User, |
| Flow of Events | 1. User decides to have Avatar travel to the past.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey.  3. The Game Client displays a prompt requesting a date.  4. User types in a date that is later than the current in-game date.  5. The Game Client, recognizing that the entered date is greater than the current date but is greater than the Set Maxed Date, displays an error message stating that the desired year cannot be reached under any circumstances. |
| Entry Condition | The User has started the game and has either loaded or initialized a world to play in. |
| Exit Condition | The User’s Avatar remains at their current date and can perform any other actions that are available at their current location at that point in history.  > The year the User wishes to go is > Maxed Set Date. |
| Quality  Requirements |  |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Traveling To an Changed Future that is Flagged “Unreachable” and is Earlier Then “Set Maxed Date” |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar travel to the future.  2. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey.  3. The Game Client displays a prompt requesting a date.  4. User types in a date that is later than the current in-game date.  5. The Game Client, recognizing that the entered date is later than the current date but less than the Set Maxed Date, checks to see if any of the years between the current date and the entered date have been flagged with ‘changed’. The check reveals that one or more years have been flagged.  6. Beginning with the first flagged year and ending with the entered date, the Game Client performs the following actions for each year in order from ‘i’ to ‘vi’:  --------i) Checks to see if flagged year is also flagged ‘Unreachable’. Once a year so flagged is found immediately proceed to step 7, skipping the other actions for this year and any years not yet reached.  --------ii) Re-calculates the state of the world for the flagged year using the information of the previous year.  --------iii) Attempts to apply any stored User Actions for the flagged year to the state of the world in the flagged year. If the preconditions of the stored User Action are not meant (such as redirecting a river that no longer exists), then it is skipped.  -------iv) The year’s ‘changed’ flag is set to ‘false’.  -------v) The next year’s ‘changed’ flag is set to ‘true’.  -------vi) Display an error message stating that the current year is not reachable. The exact contents of the error message are drawn from the parameter specification. If the parameters do not specify an error message, the error message simply states that the current year is unreachable. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User has performed one or more actions that has caused the game to flag a year between their current year and the one they wish to go to (including the one they wish to go to) with ‘changed’. |
| Exit Condition | > The User’s Avatar is remains at the current date and can perform any other actions that are available at their current location at that point in history.  > All reachable years between the current date and the year flagged as “unreachable’ have been updated and are no longer flagged as “changed”. |
| Quality  Requirements | This process should not take longer than 40 seconds. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Acquire Object (Successfully) |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar acquire an object.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the object.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the User’s available actions and displays them along with a number corresponding to each one. One of the displayed actions is “Acquire <Object Name>”  5. The User inputs the number of “Aquire <Object Name>.  6. The Game Client checks the parameter lists to see if there is a limitation on inventory space and, if so, compares the size of the object (also drawn from parameter lists) to the amount of remaining inventory space. The Game Client confirms that there is enough space remaining.  7. The Game Client checks the parameter lists to see if there are any special restrictions involved with acquiring this type of object that would be violated by acquiring it. The Game Client does not find any such restrictions.  8. The object is added to the User’s inventory. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square where there is an object that they can aquire.  >The User’s inventory has open space. |
| Exit Condition | > The User’s inventory now contains the acquired object. |
| Quality  Requirements | Some objects might have incomplete parameters, so the game should have built-in standard settings (infinite space and all objects size 1) in case the parameter lists don’t specify anything. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Acquire Object (Inventory is Full) |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar acquire an object.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the object.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the User’s available actions and displays them along with a number corresponding to each one. One of the displayed actions is “Acquire <Object Name>”  5. The User inputs the number of “Aquire <Object Name>.  6. The Game Client checks the parameter lists to see if there is a limitation on inventory space and, if so, compares the size of the object (also drawn from parameter lists) to the amount of remaining inventory space. The Game Client confirms that there is not enough space remaining.  7. The Game Client displays an error message informing the User that there is not enough space remaining in their inventory. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square where there is an object that they can aquire.  >The User’s inventory has insufficient open space. |
| Exit Condition | > The User’s inventory remains unchanged.  > The User can otherwise continue to perform any available actions at their Avatar’s current location at its current date. |
| Quality  Requirements | Some objects might have incomplete parameters, so the game should have built-in standard settings (all objects size 1) in case the parameter lists fail to specify the needed inventory information. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Acquire Object (Special Restrictions) |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar acquire an object.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the object.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the User’s available actions and displays them along with a number corresponding to each one. One of the displayed actions is “Acquire <Object Name>”  5. The User inputs the number of “Aquire <Object Name>”.  6. The Game Client checks the parameter lists to see if there is a limitation on inventory space and, if so, compares the size of the object (also drawn from parameter lists) to the amount of remaining inventory space. The Game Client confirms that there is enough space remaining.  7. The Game Client checks the parameter lists to see if there are any special restrictions involved with acquiring this type of object that would be violated by acquiring the object. The Game Client finds one such restriction.  8. The Game Client displays an error message whose contents are drawn from in the parameter list. If the parameter list does not specify a message, a “Cannot acquire item due to special restrictions” message is displayed. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square where there is an object that they can acquire.  >The User’s inventory has sufficient open space.  > There exists a special restriction in the parameter lists that disallows the User from acquiring that item at this point in time. |
| Exit Condition | > The User’s inventory remains unchanged.  > The User can otherwise continue to perform any available actions at their Avatar’s current location at its current date. |
| Quality  Requirements | Some objects might have incomplete parameters, so the game should have built-in standard settings (infinite inventory space and all objects size 1) in case the parameter lists fail to specify the needed inventory information. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Use Object |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar use an Object in their inventory.  2. User either navigates the menu (pressing ‘5’ for ‘inventory’ to bring up a list of all objects in the User’s inventory along with their associated number or presses a pre-set hotkey to do the same. The User then presses the number corresponding to the Object they would like to use.  3. The Game Client checks the User’s inventory and the parameters for this world to see what ways the User can use the Object.  4. The Game Client displays the User’s available uses for the Object and displays them along with a number corresponding to each one.  5. The User inputs the number of the method of use they prefer.  6. The Game Client checks the parameter lists to determine the results of that use of the object and sends the appropriate input to the World Map, NPC Manager, Inventory, and Short Term History Storage.  7. The sent input is used to determine the new state of the game world. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s inventory contains an object. |
| Exit Condition | > The game is updated with the results of using the object and the next year has been flagged as “changed”. |
| Quality  Requirements | “Do Nothing” should always be an option, even if it not in the parameter lists. Furthermore, a use method whose results are not defined in the parameter lists should also be treated as “Do Nothing”. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Remove Object From Inventory |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar remove an Object from their inventory.  2. User either navigates the menu (pressing ‘5’ for ‘inventory’ to bring up a list of all objects in the User’s inventory along with their associated number or presses a pre-set hotkey to do the same. The User then presses the number corresponding to the Object they would like to use.  3. The Game Client checks the User’s inventory and the parameters for this world to see what ways the User can use the Object.  4. The Game Client displays the User’s available uses for the Object and displays them along with a number corresponding to each one.  5. The User inputs the number corresponding to “dropping” the object.  6. The Game Client checks the parameter lists to determine the results of that use of the object and sends the appropriate input to the World Map, NPC Manager, Inventory, and Short Term History Storage.  7. The sent input is used to determine the new state of the game world. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s inventory contains an object.  > Removing/dropping the object is listed as a possible use of the object in the parameter lists. |
| Exit Condition | > The game is updated with the results of removing the object from the User’s inventory. This will usually just be a less full inventory, but some objects might do special things when discarded (nuclear waste?). |
| Quality  Requirements | There should be a preset “remove” use method that can be easily set for all objects when a developer (third party or otherwise) is creating a parameter list. The official starting parameter lists and the example template for third party use should come with this method set in order to make it the usual way objects work unless actively changed. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Craft Object (With Inventory Space) |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar remove an Object from their inventory.  2. User either navigates the menu (pressing ‘5’ for ‘inventory’ to bring up a list of all objects in the User’s inventory along with their associated number or presses a pre-set hotkey to do the same. The User then presses the number corresponding to the Object they would like to use.  3. The Game Client checks the User’s inventory and the parameters for this world to see what ways the User can use the Object.  4. The Game Client displays the User’s available uses for the Object and displays them along with a number corresponding to each one.  5. The User inputs the number corresponding to “Crafting” with the object  6. The Game Client checks the User’s inventory and the parameters for this world to see if there are any other objects in the User’s inventory that can be crafted along with this object.  7. The Game Client displays all objects along with a corresponding number that can be used with the chosen object to create a new object. The chosen object itself might be an option (I.E. the object can be crafted into something else without needing other recourses.)  8. The User inputs a number corresponding to additional object they would like to use.  9. The Game Client checks the parameter lists to determine the results of crafting the two items together and updates the User’s inventory accordingly. It then checks to see if there will be enough inventory space after all inventory changes are made. It confirms there is enough space.  10. The Game Client alters the User’s inventory according to its calculations. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s inventory contains two or more objects that can be used together via the “crafting” use method.  > The User has enough open inventory space to store the item they wish to craft. |
| Exit Condition | > The User’s inventory is updated with the results of the crafting action. |
| Quality  Requirements | > It should not be assumed that crafting uses up an item, although there should be a preset “crafting consumes object” use method that can be easily set for all objects when a developer (third party or otherwise) is creating a parameter list to allow a developer to easily set such a standard.  > Some objects might have incomplete parameters, so the game should have built-in standard settings (infinite inventory space and all objects size 1) in case the parameter lists fail to specify the needed inventory information. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Craft Object (Without Inventory space) |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar remove an Object from their inventory.  2. User either navigates the menu (pressing ‘5’ for ‘inventory’ to bring up a list of all objects in the User’s inventory along with their associated number or presses a pre-set hotkey to do the same. The User then presses the number corresponding to the Object they would like to use.  3. The Game Client checks the User’s inventory and the parameters for this world to see what ways the User can use the Object.  4. The Game Client displays the User’s available uses for the Object and displays them along with a number corresponding to each one.  5. The User inputs the number corresponding to “Crafting” with the object  6. The Game Client checks the User’s inventory and the parameters for this world to see if there are any other objects in the User’s inventory that can be crafted along with this object.  7. The Game Client displays all objects along with a corresponding number that can be used with the chosen object to create a new object. The chosen object itself might be an option (I.E. the object can be crafted into something else without needing other recourses.)  8. The User inputs a number corresponding to additional object they would like to use.  9. The Game Client checks the parameter lists to determine the results of crafting the two items together and updates the User’s inventory accordingly. It then checks to see if there will be enough inventory space after all inventory changes are made. It confirms there is not enough space.  10. The Game Client displays an error message to the User stating that there is not enough space for the item that is to be crafted. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s inventory contains two or more objects that can be used together via the “crafting” use method.  > The User does not have enough open inventory space to store the item they wish to craft. |
| Exit Condition | > The User’s inventory remains unchanged. |
| Quality  Requirements | > Some objects might have incomplete parameters, so the game should have built-in standard settings (all objects size 1) in case the parameter lists fail to specify the needed inventory information. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Interact with Object |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar interact with an object present in their current square.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the object.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the Users available actions and displays them along with a number corresponding to each one.  5. The User inputs the number of the action they would like to take.  6. The Game Client checks the parameters to determine the results of the action.  6. The Game Client checks the parameter lists to determine the results of interacting with the chosen object in the chosen way and sends the appropriate input to the World Map, NPC Manager, Inventory, and/or Short Term History Storage as needed.  7. The sent input is used to determine the new state of the game world. |
| Entry Condition | > The User’s has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square that contains an object that can be interacted with. |
| Exit Condition | > The World Map, Short Term History, Inventory, and/or NPC instances are modified in accordance to the results of interacting with the object and the next year has been flagged as “changed”. The User’s Avatar remains at its current location and date and can perform any actions available at that current location and date. |
| Quality  Requirements | Input should only be sent to parts of the program for which the interaction is relevant. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Interact with Object |
| Participating Actors | User |
| Flow  of Events | 1. User decides to have their Avatar interact with an object present in their current square.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the geographical feature) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the geographical feature.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the Users available actions and displays them along with a number corresponding to each one.  5. The User inputs the number of the action they would like to take.  6. The Game Client checks the parameters to determine the results of the action.  6. The Game Client checks the parameter lists to determine the results of interacting with the chosen geographical feature in the chosen way and sends the appropriate input to the World Map, NPC Manager, Inventory, and/or Short Term History Storage as needed.  7. The sent input is used to determine the new state of the game world. |
| Entry Condition | > The User’s has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square that contains a geographical feature that can be interacted with. |
| Exit Condition | > The World Map, Short Term History, Inventory, and/or NPC instances are modified in accordance to the results of interacting with the geographical feature and the next year has been flagged as “changed”. The User’s Avatar remains at its current location and date and can perform any actions available at that current location and date. |
| Quality  Requirements | Input should only be sent to parts of the program for whom the interaction is relevant (although most alterations in geography will have wide-spread effects) |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Interact With a Community |
| Participating Actors | User |
| Flow  of Events | 1. User decides to have their Avatar interact with the community located at its current location.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the community) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the community.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the Users available actions and displays them along with a number corresponding to each one.  5. The User inputs the number of the action they would like to take.  6. The Game Client checks the parameters to determine what input to send to the NPC instance.  7. The Game Client works through the NPC manager to send the calculated input to the NPC instance corresponding to the community.  8. The NPC instance updates itself in accordance to the given input.  9. The next year is flagged as “changed”. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square on the world map where a community/part of a community is located. |
| Exit Condition | > The Community has changed in accordance to the action and the next year has been flagged as “changed”. The User’s Avatar remains at its current location and date and can perform any actions available at that current location and date. |
| Quality  Requirements |  |

|  |  |
| --- | --- |
| Use Case Name | Playing Game – Interact With an Individual |
| Participating Actors | User |
| Flow of Events | 1. User decides to have their Avatar interact with the community located at its current location.  2. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the individual) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the individual.  3. The Game Client checks the User’s inventory and the parameters for this world to see what actions the User has available.  4. The Game Client displays the Users available actions and displays them along with a number corresponding to each one.  5. The User inputs the number of the action they would like to take.  6. The Game Client checks the parameters to determine what input to send to the NPC instance.  7. The Game Client works through the NPC manager to send the calculated input to the NPC instance corresponding to the individual.  8. The NPC instance updates itself in accordance to the given input.  9. The next year is flagged as “changed”. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in.  > The User’s Avatar is in a square on the world map where an individual NPC that can be interacted with is located (possibly within a community). |
| Exit Condition | > The individual has changed in accordance to the action and the next year has been flagged as “changed”. The User’s Avatar remains at its current location and date and can perform any actions available at that current location and date. |
| Quality  Requirements |  |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Using Local Scan |
| Participating Actors | User |
| Flow  of Events | 1. User decides to have their Avatar interact with the community located at its current location.  2. User either navigates the menu (pressing ‘2’ for ‘Local Scan’ to bring up a list of all information groups and then pressing the number corresponding to the desired group) or presses a previously set hotkey to bring up the list of all scanable information groups and then pressing the the number corresponding to the one they desire to learn about.  3. The Game Client consults the Short Term History, the World Map, and (if an NPC is present in the square) the NPC manager to gather knowledge of the specified type that pertains to the current contents of the square the User’s Avatar is currently in.  4. The Game Client displays a summary of the data to the User. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in. |
| Exit Condition | > The game world is unchanged and the User knows more about the contents of the square their Avatar is currently in. |
| Quality  Requirements | The default information groups should be Full, Society, and Nature and the instructional parameter list template should explain how to create additional information groups. |

|  |  |
| --- | --- |
| Use Case Name | Playing Game – Using Global Scan |
| Participating Actors | User |
| Flow  of Events | 1. User decides to have their Avatar interact with the community located at its current location.  2. User either navigates the menu (pressing ‘3’ for ‘Global Scan’ to bring up a list of all information groups and then pressing the number corresponding to the desired group) or presses a previously set hotkey to bring up the list of all scanable information groups and then pressing the the number corresponding to the one they desire to learn about.  3. The Game Client consults the Short Term History, the World Map, and (if there are NPCs present in the world at the Avatar’s current Date) the NPC manager to gather knowledge of the specified type that pertains to the point in history the User’s Avatar is currently in.  4. The Game Client displays a summary of the data to the User. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in. |
| Exit Condition | > The game world is unchanged and the User knows more about the state of the world at their Avatar’s present point in history. |
| Quality  Requirements | The default information groups should be Full, Society, and Nature and the instructional parameter list template should explain how to create additional information groups. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game – Confirm Victory |
| Participating Actors | User |
| Flow of Events | 1. User decides to confirm their victory.  2. User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Confirm Victory’) or presses a previously set hotkey and then the number corresponding to ‘Confirm Victory’.  3. The Game Client consults the Short Term History, the parameter lists, and the NPC manager (if needed) to determine if the User meets the victory conditions. The User meets the conditions.  4. The Game Client consults the Short Term History, the parameter lists, and the NPC manager (if needed) to determine the User’s Score.  5. The Game Client compares the User’s Score with their High Score for this particular world. If it is greater, the new score replaces the old score.  6. The Game Client displays a prompt congratulating the player.  7. The Game world is otherwise unchanged. |
| Entry Condition | > The User has started the game and has either loaded or initialized a world to play in. |
| Exit Condition | > The User’s High Score for this particular world is now the larger of their previous High Score and the score of their current victory.  > The game world is otherwise unchanged (it might be possible to change history in a way that wins in an even more impressive fashion.) |
| Quality  Requirements | The default High Score should be set to 0. If a parameter list does not detail a way to win, the standard score system will reward 1 point per year civilization exists. |

|  |  |
| --- | --- |
| Use Case Name | Playing Game - Load Game |
| Participating Actors | User |
| Flow of Events | 1.    The User wishes to load a previously saved game.  2.    User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Load Game’) or presses a previously set hotkey and then the number corresponding to ‘Load Game’.  3. The Game Client will consult its list of Long Term Storage files to display a list of saved game sessions.  4. The User will enter the number corresponding to a desired game session.  5. The Game Client will load the world, inventory, NPCs, and Avatar location detailed in the Long Term Storage file for the desired game session.  6.    A confirmation of a successful game load will be given to the User, the game play screen will resume. |
| Entry Condition | > The User wishes to play a previously saved game. |
| Exit Condition | > The User receives confirmation that his/her game has been successfully loaded and game play resumes. |
| Quality  Requirements | The game should display a loading bar and the entire process should not take longer than two minutes. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Save Game |
| Participating Actors | User |
| Flow of Events | 1.    The User wishes to save their current game session game.  2.    User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Save Game’) or presses a previously set hotkey and then the number corresponding to ‘Save Game’.  3. The Game Client displays a prompt asking the User to name this session.  4. The Game Client stores the World Map, Inventory, NPC, and Short Term History data in this sessions in a Long Term Storage file with the specified name. If this would overwrite a pre-existing file the User is asked to confirm the process. The User confirms.  5.    A confirmation of a successful save will be given to the User, the game play screen will resume. |
| Entry Condition | > The User wishes to play a previously saved game. |
| Exit Condition | > The User receives confirmation that his/her game has been successfully loaded and game play resumes. |
| Quality  Requirements | The game should display a loading bar and the entire process should not take longer than 30 seconds. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Set Hot Key |
| Participating Actors | User |
| Flow of Events | 1.    The User wishes to set new hot keys.  2.    User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Set Hot Keys’) or presses a previously set hotkey and then the number corresponding to ‘Set Hot Key’.  3. The Game Client displays a list of Hot Keys, each with a corresponding number, and their current key binding.  4. The User presses a number corresponding with a hot key and then a new key that becomes bound to the previous hot key. This step can be repeated any number of times until the User presses the number corresponding with ‘return to game’. |
| Entry Condition | > The User wishes to change the hot key bindings. |
| Exit Condition | > The User’s hot key bindings for the game have been changed. |
| Quality  Requirements | The User should not be able to set a number as a hot key. |

|  |  |
| --- | --- |
| Use  Case Name | Playing Game - Start New Game |
| Participating Actors | User |
| Flow of Events | 1.    The User wishes to start a new game.  2.    User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘New Game’) or presses a previously set hotkey and then the number corresponding to ‘New Game’.  3. The Game Client displays a prompt asking the User if they would like to save their current session. If the User confirms, the current session is saved.  4. The Game Client displays a prompt asking the User to choose a parameter file set for their new game session.  5. The User scrolls down a list and clicks on the desired parameter file set.  6. The Game Client generates a new World Map, inventory, NPC manager, new NPC instances, Long Term Storage and Short Term History Storage classes/files for the new game world in accordance with the selected parameter list(s)  7.    The User’s Avatar begins at a location and date specified in the parameter files, or at year 0 in the bottom-left corner if not specified. |
| Entry Condition | > The User wishes to start a new game. |
| Exit Condition | > The User is now in a new game session. |
| Quality  Requirements | The game should display a loading bar and the entire process should not take longer than three minutes. |

|  |  |
| --- | --- |
| Use Case Name | Playing Game - Quit Game |
| Participating Actors | User |
| Flow of Events | 1.    The User wishes to quite the game  2.    User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Quit’) or presses a previously set hotkey and then the number corresponding to ‘Quit’.  3. The Game Client displays a prompt asking the User if they would like to save their current session. If the User confirms, the current session is saved.  4. The Game Client exits the current game session and ceases to run. |
| Entry Condition | > The User wishes to quit the game. |
| Exit Condition | > The Game ends and the User exits the program. |
| Quality  Requirements |  |

**Use Cases Beyond Simply Playing the Game**

|  |  |
| --- | --- |
| Use Case Name | Update Game Core |
| Participating Actors | Development Team, Website |
| Flow of Events | 1.    Development team identifies bugs that cause the application from running correctly according to crash data provided by the software.  2.    Development team updates appropriate code.  3.    Development team replaces existing software installation files on the website. |
| Entry Condition | User crash data submittals after application has crashed on the User end. |
| Exit Condition | Crash data (bugs) have been fixed and a new version has been uploaded to the website for User download. |

|  |  |
| --- | --- |
| Use Case Name | Create Parameter File(s) |
| Participating Actors | Development Team, Company, Company Website |
| Flow of Events | 1.    Development team and/or User desires to alter the application beyond the original game play experience by creating a new set of Parameter Lists.  2.    Development team and/or User will design the new game file(s) (.gdi) such that it reflects the designers desired game play experience.  3.    Development team and/or User (after extensive play testing) can then upload their creations to the game extensions server on the company website.  4.    The game extension file can then be downloaded by other Users interested in playing that particular game file. |
| Entry Condition | Development team and/or User wishes to add additional world parameters or create entire parameter sets for use with the game. |
| Exit Condition | Development team and/or User has uploaded their game extension file to the game extensions server. |

|  |  |
| --- | --- |
| Use Case Name | Download Parameter Files |
| Participating Actors | User, Website |
| Flow of Events | 1.    The User wishes to supplement the game with developer and User submitted parameter files.  2. The User logs into the company website and clicks the button labled “Downloads”  3. The User selects the desired parameter file set.  4. The parameter files are saved in the appropriate folders. |
| Entry Condition | The User wants to play and attempt User submitted game files. |
| Exit Condition | The User has successfully downloaded the new parameter file(s). |

|  |  |
| --- | --- |
| Use Case Name | View High Scores |
| Participating Actors | User, Website |
| Flow of Events | 1. User wishes to see the high scores.  2. User goes onto the website.  3. User navigates to the “high scores” page. |
| Entry Condition | The User has a functional internet connection. |
| Exit Condition | The User has successfully viewed the high scores |
| Quality Requirements | Finding the High Scores should take no longer than 5 minutes for the average user. |

|  |  |
| --- | --- |
| Use Case Name | Submit High Scores |
| Participating Actors | User, Website |
| Flow of Events | 1. User wishes to submit their high scores.  2. User goes onto the website.  3. User navigates to the “high scores” page.  4. The user hits the “submit high scores” button.  5. The website displays a prompt asking the user browse their computer for a long-term storage file to draw the high score from.  6. The website extracts the high score and adds it to their data base if it is within the top 10,000. |
| Entry Condition | The User has a functional internet connection. |
| Exit Condition | The User has submitted their high score. |
| Quality Requirements | Submitting High Scores should take no longer than 6 minutes for the average user. |

|  |  |
| --- | --- |
| Use Case Name | Create User Account |
| Participating Actors | User, Website |
| Flow of Events | 1. User wishes to create a user account on the website.  2. User goes onto the website.  3. User navigates to the “create an account” page.  4. User completes all the required information to create an account. |
| Entry Condition | The User has a functional internet connection. |
| Exit Condition | The User has successfully created an Account |
| Quality Requirements | Creating an account should take no longer than 7 minutes for the average user. |

|  |  |
| --- | --- |
| Use Case Name | Download Game |
| Participating Actors | User, Website |
| Flow of Events | 1. User wants to download the game.  2. User goes onto the website.  3. User logs into their account.  4. User navigates to the “downloads” page.  5. User selects the version of the game for their operating system.  6. The game is downloaded onto the User’s computer. |
| Entry Condition | The User has a functional internet connection. |
| Exit Condition | The User has successfully downloads the game |
| Quality Requirements | Downloading the game should take no longer than 10 minutes for the average user. |

|  |  |
| --- | --- |
| Use Case Name | Install Game |
| Participating Actors | User |
| Flow of Events | 1. User runs the program that was downloaded from the website (for windows this will be a .exe file)  2. The program guides the user through the installation process.  3. The game is installed onto the users system and a shortcut is created on the user’s desktop. |
| Entry Condition | The User has downloaded the game. |
| Exit Condition | The User has successfully installed the game onto their system. |
| Quality Requirements | Installing the game should take no longer than 10 minutes for the average user. |

## Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 1 |  | Requirement Type: | 10 |
| Description: | The application must install successfully on the desired user platform (operating system). | | |
| Rationale: | The user should not be expected to encounter or deal with errors involving installation on his/her desired operating system. | | |
| Originator: |  | | |
| Fit Criterion: | Can the application be installed on the user’s operating system of choice? | | |
|  |  |  |  |
| Customer Satisfaction: | 5 | Customer Dissatisfaction: | 5 |
| Priority: | 5 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 12/06/2014 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 2 |  | Requirement Type: | 10 |
| Description: | The application must display the main menu interface upon start-up and user invocation. | | |
| Rationale: | The user should expect to see the main menu interface when desired. | | |
| Originator: |  | | |
| Fit Criterion: | Is the main menu interface displayed upon application launch and user invocation? | | |
|  |  |  |  |
| Customer Satisfaction: | 5 | Customer Dissatisfaction: | 5 |
| Priority: | 5 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 12/06/2014 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 3 |  | Requirement Type: | 10 |
| Description: | The application must provide a method of saving current game status for later use (save files). | | |
| Rationale: | The user should not be expected to start the game from scratch every time he/she plays the game. | | |
| Originator: |  | | |
| Fit Criterion: | Can the game create save files? Can the game load a saved game? The answer to both needs to be yes! | | |
|  |  |  |  |
| Customer Satisfaction: | 5 | Customer Dissatisfaction: | 5 |
| Priority: | 5 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 10/18/2014 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 4 |  | Requirement Type: | 10 |
| Description: | The application must provide a way to deploy user-created parameter files to allow the game to be played with different parameter sets. | | |
| Rationale: | Being able to change the parameter sets would greatly enhance the longevity of the game, changing our game from a single game to a vast library of similar titles. | | |
| Originator: |  | | |
| Fit Criterion: | Can 100% of a series of test subjects of various ages and backgrounds navigate our website, load a user-created parameter file, and begin a game session using that file in under 25 minutes? And can at least 60% of them do it in under 15? | | |
|  |  |  |  |
| Customer Satisfaction: | 5 | Customer Dissatisfaction: | 2 |
| Priority: | 4 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 10/18/2014 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 5 |  | Requirement Type: | 10 |
| Description: | The application must support crash error data submissions. | | |
| Rationale: | The user must have the choice to submit crash data upon an application crash. These crash data submittals will assist the development team in improving and fixing code errors. | | |
| Originator: |  | | |
| Fit Criterion: | We should run a gauntlet of various crash types (not enough memory, power outage, etc) and check each time to see if the User is given the option to submit crash data (and that the data is submitted properly). The program should have a success rate of at least 80% | | |
|  |  |  |  |
| Customer Satisfaction: | 1 | Customer Dissatisfaction: | 3 |
| Priority: | 3 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 10/18/2014 | | |

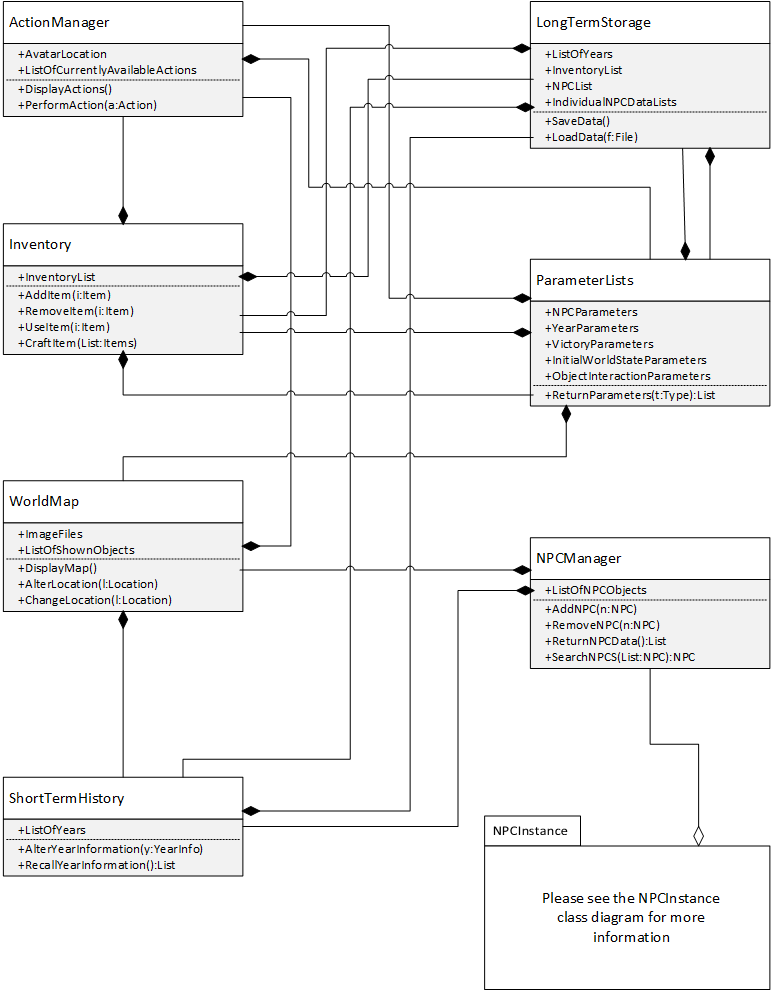
|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 6 |  | Requirement Type: | 10 |
| Description: | The application must have a system of viewing previously submitted high scores. This data is retrieved from the high score server. | | |
| Rationale: | The user will want information regarding where their high scores rank amongst the high scores of all collective players. | | |
| Originator: |  | | |
| Fit Criterion: | Can 100% of a series of a test subjects navigate our website and find the high scores section in under 5 minutes? And can 60% of them do it in under 3? | | |
|  |  |  |  |
| Customer Satisfaction: | 2 | Customer Dissatisfaction: | 1 |
| Priority: | 2 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 10/18/2014 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 7 |  | Requirement Type: | 10 |
| Description: | The application must perform every action as signaled from the main menu interface and in game commands. | | |
| Rationale: | The application must respond to all valid user input to perform the desired action. | | |
| Originator: |  | | |
| Fit Criterion: | Are the actions during game play the direct result of user input? | | |
|  |  |  |  |
| Customer Satisfaction: | 2 | Customer Dissatisfaction: | 1 |
| Priority: | 2 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 12/06/2014 | | |

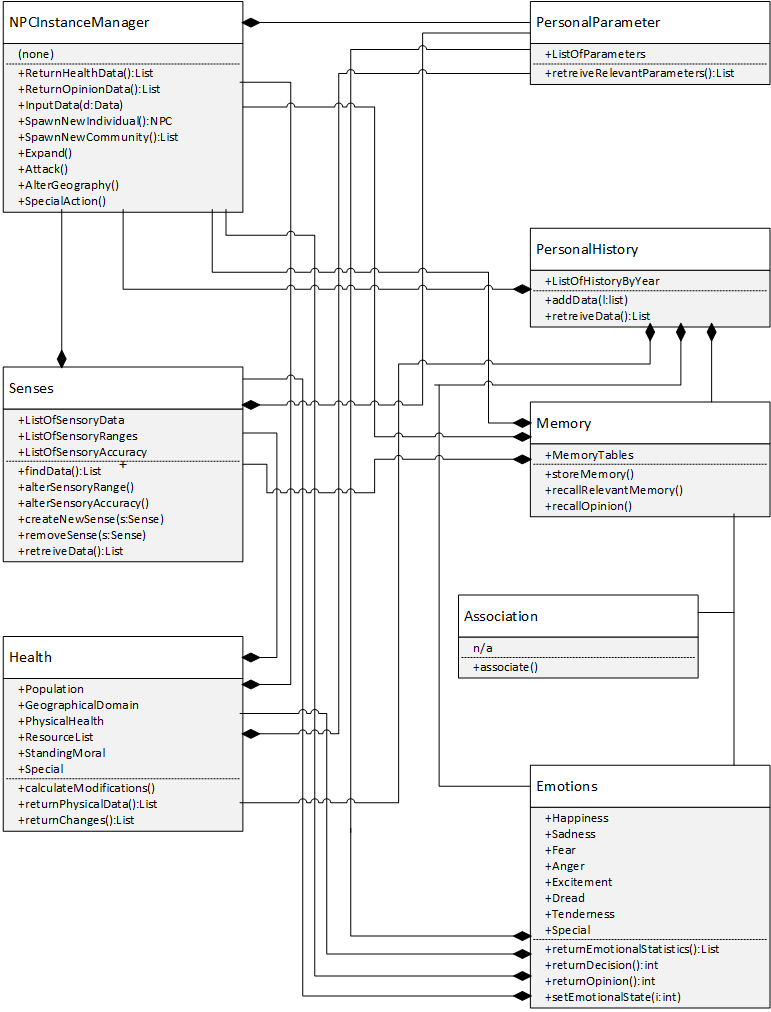
|  |  |  |  |
| --- | --- | --- | --- |
| Requirement #: 8 |  | Requirement Type: | 10 |
| Description: | The NPC (non-player character) architecture must be capable of handling both individual NPC actions and community/civilization NPC actions. | | |
| Rationale: | The NPC architecture must be scalable to handle both individual and community/civilization rationale and actions and reactions. | | |
| Originator: |  | | |
| Fit Criterion: | Does the NPC architecture handle both individual and community/civilization AI (artificial intelligence)? | | |
|  |  |  |  |
| Customer Satisfaction: | 2 | Customer Dissatisfaction: | 1 |
| Priority: | 2 | Conflicts: |  |
| Supporting Materials: |  |  |  |
| History: | Last Updated October 12/06/2014 | | |

## Data Requirements

Overall game class diagram:



NPC Instance class diagram:



## Performance Requirements

### Speed and Latency Requirements

When the user advances to the next “game” screen, it should not take more than 0.000005 seconds to render.

When the user switches from one “age” to another “age” in time in the game, the loading times should not exceed 50 seconds.

The application should not take more than 2 seconds to query the database of next possible moves available to the user.

The user should not be able to notice any latency when picking up and attaining inventory items throughout the game.

### Precision or Accuracy Requirements

When users develop additional game files to “extend” the core game maps, the map game file (.gdi), must be written with the appropriate file syntax as specified by game format information file specification.

### Capacity Requirements

The end–user’s machine, or computer, must have enough available space to accommodate the game installation and any additional game extension files that are either created or downloaded from the game extension server.

## Dependability Requirements

### Reliability Requirements

In the event of a game crash, the crash report data shall be sent to the crash data server and the appropriate fixes shall take place within three days by the development team. Once the game is “patched”, the development team will push the fixed version to the website and notify users that a new version is available for download that addresses the errors fixed in that particular release.

### Availability Requirements

The main website (server) should available 24-hours a day, 365 days a year in the case that new user registers for and pays for the product. The availability will allow the users to download and install the game software at any time.

The game extensions, high scores, and crash data servers should have an uptime of 98%.

### Safety-Critical Requirements

The game should not be played for more than 42 consecutive hours. Studies have shown that playing for more than 42 consecutive hours can/will cause sleep deprivation and other medical conditions.

## Maintainability and Supportability Requirements

### Maintenance Requirements

Any update to the original game file should not take more than ten minutes to download and update if the user has at least 25mb/s download speed provided by their Internet provider.

The application shall be able to notify the user if new game extension file have been uploaded.

The user will be notified of any updates to the core gaming file and core application upon running the application and the software will begin to update itself.

### Supportability Requirements

The company website shall an online help system to assist the user in registering, downloading, and installing the application.

The company website shall have a help system in place to assist the user in learning the gaming commands.

The company website shall have a help system in place to assist the user in creating game extension files.

### Adaptability Requirements

The application is expected to run under at least Windows 7, Windows 8.x, Linux, and iOS operating systems and desktop environments.

### Scalability or Extensibility Requirements

Although the “core” application only requires 80 Mbs of space upon the initial install, the extent of scalability will be determined by the user and the amount of downloadable game extensions in his/her library of downloads.

### Longevity Requirements

The application is expected to run on current operating systems available at the time of the documentation publication. In the event of major operating system updates, the software is not guaranteed to run.

## Security Requirements

### Access Requirements

The core application is only accessible by the developers (closed source).

The application game extension files, since they can be created by anyone, are open source and freely available to all who wants it.

The application high scores service (server) will be accessible to anyone from the high scores portion of the website regardless of owning the core game application or not.

### Integrity Requirements

Any user or developer wishing to create game extension files must follow the rules and syntax of the game information format (.gdi). The core application is expecting proper and correct input from the game extension file. If the game extension file is in an incorrect format, the core application will determine this and simply display a message (not necessarily exactly what the problem is).

### Privacy Requirements

Any user not wishing to share his/her username in the high scores service (server) can chose at end-game to not upload their high score.

### Audit Requirements

The application as whole (including services and servers) will track usernames (not necessarily users themselves) of anyone wishing to submit their high score to the high scores service (server).

### Immunity Requirements

The application, upon installation, shall prompt and inform the user that the application does communicate via the internet to occasionally check for software updates, game extension files, and high scores.

## Usability and Humanity Requirements

### Ease of Use Requirements

● Efficiency of use: The gaming application should take the average end user less then twenty minutes to understand how to play and the commands to use.

● Ease of remembering: The casual user will need to remember a list of about ten commands to be able to use/play the game at anytime.

● Error rates: The error rates of the end user should be less than two for every ten moves they do.

● Overall satisfaction in using the product: Overall satisfaction of using application among the test panel users should be 95%.

### Personalization and Internationalization Requirements

The application will only be offered in the English (US) locale at the time of writing of this documentation.

If the feedback from users request optional locale support, future core application updates from development may include language preferences that the user can select.

### Learning Requirements

The application will be and should be easy to learn for any user who has experience with similar gaming applications and should not be difficult even for users who do not have any experience with gaming applications. A test panel of users with various degrees of technical familiarity should be able to learn how to play the game and be able to use all user actions within 30 minutes. The test panel of users will consist of users of both genders between the ages of 10 and 50.

The built-in tutorial shall offer user assistance in getting comfortable with using the software and playing the game itself.

### Understandability and Politeness Requirements

The menu systems of the application shall be easy to identify and discoverable. A test panel of users with various degrees of technical familiarity should be able to explain how the menus work and how to perform all user actions after 40 minutes of play. The test panel of users will consist of users of both genders between the ages of 10 and 50.

Upon an application crash, the application should allow, in simple way, to have a choice to send the crash data generated by the application to the developer team and developing company. 90% of the aforementioned user test panel should be able to successfully send crash data on their first try, and 100% should be able to do so by their third try.

### Accessibility Requirements

The application will conform to the platform standards for accessibility options.

The application shall conform usage and standards for people with slighted eye sight and shall support the users of the magnification tools.

### User Documentation Requirements

Upon release or update of the core application, an included read me file in the installation folder shall contain the following: version number, release date and developer team names, etc.

The company website (in which the software is downloaded) shall have information regarding the application, such as:

* Installation procedures (from downloading, to installing, to running)
* Contact information for contacting the developing company.
* Parameter Files creation wiki (Documentation on how to create and utilize game extension files to be used with the core gaming application).

### Training Requirements

The company website shall contain a help section to assist the user in downloading and installing the application.

With regard to creating game extension files, an online wiki page shall be dedicated to this process. In the even that existing users provide video tutorials, they will be linked on these pages (in their respective area of topic). Note that user submitted videos may or may not always be available and should be considered a complete reference.

The application shall have a built-on tutorial that will assist the user in playing and learning the game concepts and theory.

## Look and Feel Requirements

### Appearance Requirements

The application (in game) shall possess a stylish graphic style similar to a semi-realistic cartoon appearance. The gaming application shall not focus on realism, instead choosing a more elegant but simplistic style of symbolic imagery.

### Style Requirements

The application shall appear to be elegant but simplistic. The game objects/NPCs/etc shall be distinctive but realistic looking art will not be stressed. Such styles stand the test of time better – early Nintendo games looks far better today than contemporary games that focuses on “realistic” graphics because such graphics are heavily overshadowed by later games.

The in-game graphical style shall appeal to all ages.

## Operational and Environmental Requirements

### Expected Physical Environment

The gaming application shall be able to be played at home, at school (on a laptop), or anywhere the player is that has an environment conducive to computing.

### Requirements for Interfacing with Adjacent Systems

The application shall be interfaced with the high score server / database, crash data server / database, and the game extensions server.

After the user has completed and / or reached an end-game scenario, the user shall have the choice of sending their respective score to the high-score server / database (if the score is deemed a high score).

The application shall generate crash data reports in the event of an application failure. The user shall have the option of sending crash data to the crash data server / database upon a failure event.

Should the user want to add additional features or maps to the gaming application, the user shall be made aware (in-game) that such options are available for download from the website.

### Productization Requirements

The application shall be available for purchase from the websites online store using an approved secure transaction method. After the user has purchased the application, the user shall be able to download the application from the website.

The application installation file(s) must not take longer than 15 minutes to download in its entirety.

All development team and/or user created game extension files (for adding additional content to the core application) shall be freely available for download from the website.

### Release Requirements

Beyond the initial core application release, the following maintenance release requirements shall be followed:

Based on crash data reports and user feedback, subsequent maintenance releases shall address these issues and be made available for download from the website.

Each maintenance release shall possess a higher quality product than the previous release and shall not cause precedent features to crash.

Each maintenance release shall include a readme file with the distribution outlining all bugs fixed in that release and list of potential known bugs (if known at the time of release).

## Cultural and Political Requirements

### Cultural Requirements

The product shall not be offensive to religious or ethnic groups. We will test this via a multicultural panel. If any member of the panel takes offense to something in the game, we will show the game to a panel consisting of solely that religious/ethnic groups to confirm it is generally offensive.

### Political Requirements

The product shall not display content that is prohibited by law in the U.S. Ports of the game to other countries must likewise abide by the laws of that country.

## Legal Requirements

### Compliance Requirements

The product shall comply with individual’s right to privacy.

The crash data shall not display any personal information from the end user it came from.

The high score data shall not show any personal information beyond the user’s profile name.

### Standards Requirements

The product shall comply with all Mélange Computing Services requirements.

# Design

## System Design

### Design goals

**Performance criteria**

**Response time:** After initial application start-up and the main menu screen is visible; the application shall not exhibit any visible delays when the user selects any available option. For instance, if a user selects to “load a saved game”, the user shall not see any visible delay or lag upon the application switching to the desired screen. Additional examples include, but are not limited to: while in-game play, the switching of map scenes shall not have visible delay; while in-game play, if the user opts to view the inventory screen, the user shall not see any visible delay or lag; while in-game, the user shall not see any visible delay when prompting the application to show the in-game menu screen (to save or load games for instance).

**Throughput:** Similarly to response time, the application shall not present any visible delay or lag when switching between various menus and screens throughout game play.

**Dependability criteria**

**Robustness:** Upon invalid user input, the application will simply do nothing. In conformance with most modern gaming applications, invalid input will simply be ignored.

**Availability:** Generally, the application is installed and run locally on the users machine and therefore shall be considered to always be available. High scores are saved per user basis on remote servers and these servers shall retain a 95 % uptime availability. Application updates and patches will be available on the company/game website and shall retain a 95% uptime availability.

**Fault tolerance:** In the event of an application crash or unrecoverable error, which results in the application crashing, the application shall offer a choice to the user to send crash data results to the development team crash data log servers.

**Security:** The application need only to communicate with high scores servers in the event of the user opting to share/send his/her high score to the high scores database server. These data transmissions shall not send any user information beyond the users chosen screen name and shall not contain any sensitive data and therefore does not require data encryption for transmission.

**Cost criteria**

**Development cost:** Development costs shall include provisions to pay design and development engineering teams according to the company standard. Initial development costs also include monthly or yearly server and hosting rates to provide high score services and game website services (where users have access to game patches and updates).

**Deployment cost:** The application is currently intended to be distributed via the company/game website as a downloadable executable installation file and therefore, beyond server and hosting fees, shall not incur and deployment costs.

**Upgrade cost:** In the event that the high scores database server sees an increase in traffic beyond the initial estimate, an increase in hosting bandwidth may be incurred.

**Maintenance cost:** The end-user of the application shall not incur any additional fees for routine maintenance (by the development teams) to the application. Maintenance fees for bug fixes and game updates and/or patches to the application will include standard operating costs and pay rates for the development team as long as the application remains profitable.

**Maintenance criteria:**

**Extensibility:** The application shall provide both the end-user and the development team the ability to create and distribute custom game maps and enhancements via the predetermined game file format.

**Portability:** The application shall be designed, such that, it will not incur a period of longer than 6 months to port to a different operating system.

**Readability:** New members to the development team(s), shall be able to understand the code and system design layout within 2 days. This implies that the application code shall be well commented and documented and shall follow the adopted design patterns.

**End user criteria**

**Utility:** The application shall provide the intended entertainment value to the end user as outlined in marketing schemes.

**Usability:** The application shall provide a simple, but short, tutorial walkthrough for first-time users that will help the user learn the controls and general game play techniques.

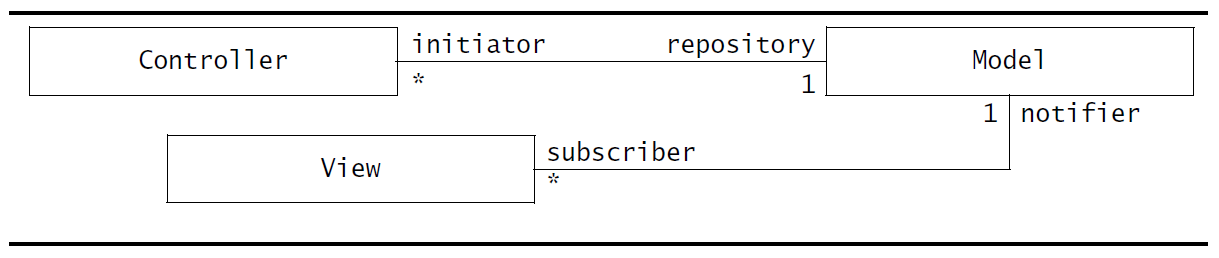
## Current Software Architecture

Temporal Architect is being designed from scratch for the client and is not meant to replace, update, or improve any existing, or current software or their corresponding architectures.

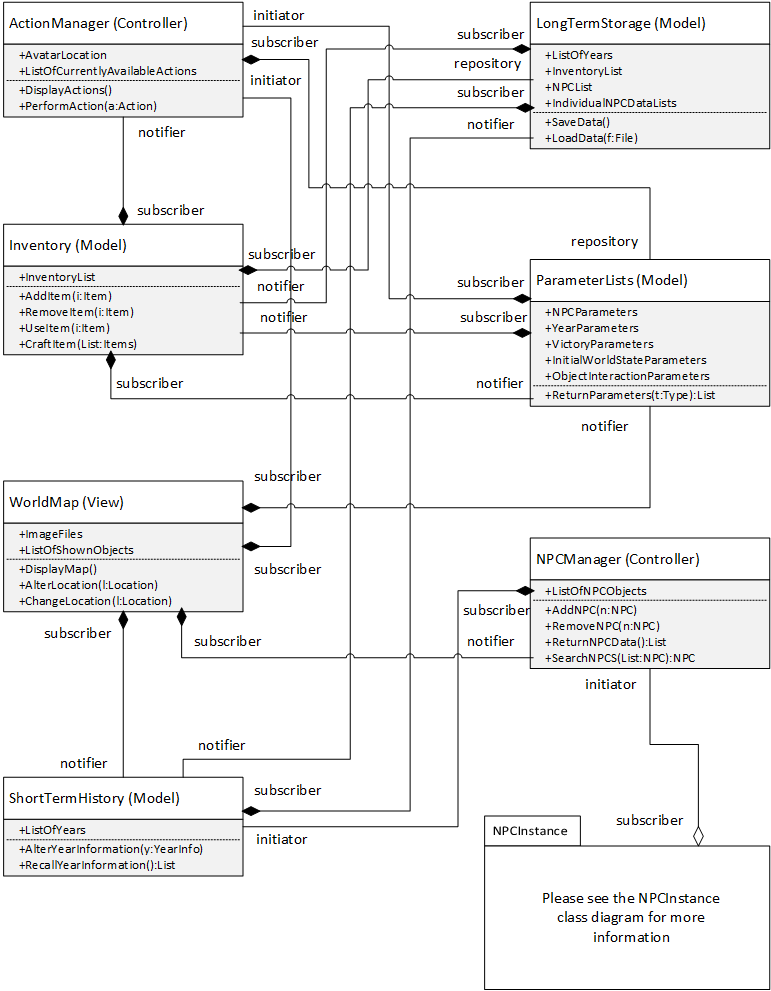
## Proposed Software Architecture

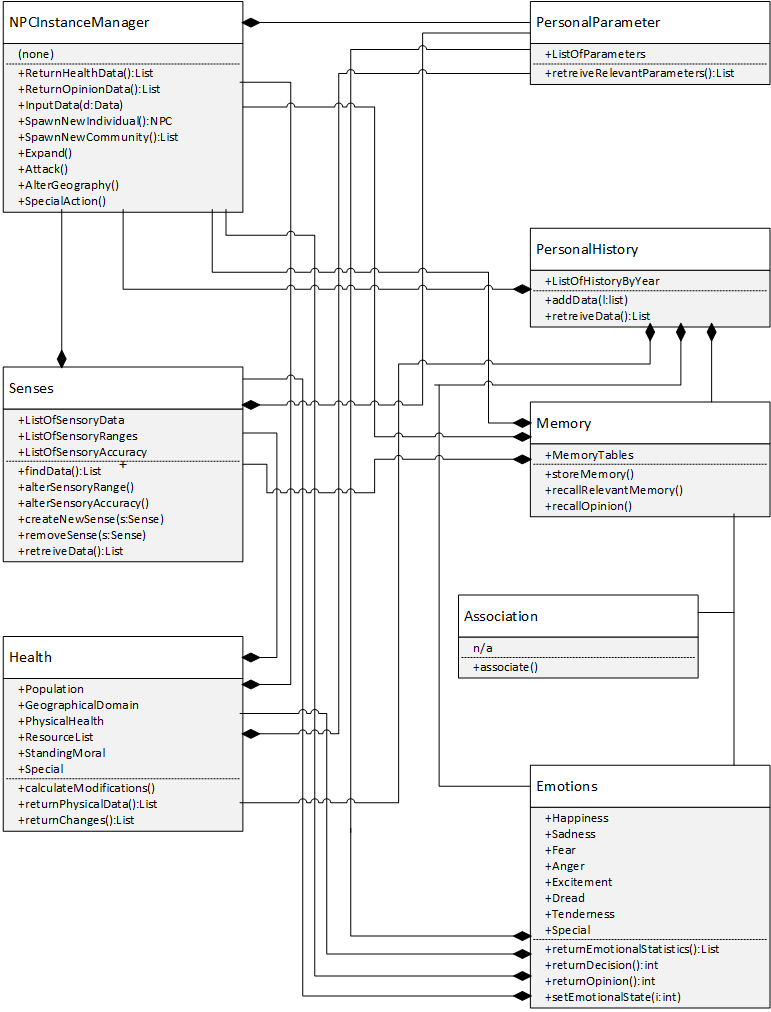
### Overview

The Temporal Architect gaming application will deploy the Model/View/Controller software architecture. The controllers will be the main interaction logic with the user such as displaying the menu screens when the user prompts the application for them. The models will maintain and generate the proper actions and responses from the users input. The models are behind-the-scenes logic, which is the main decision center of the gaming application. The view components will consists of what the user sees on the screen. For example, the world map gets updated or altered and the changes are displayed on the screen to the user. The model/view/controller design layout is as follows:



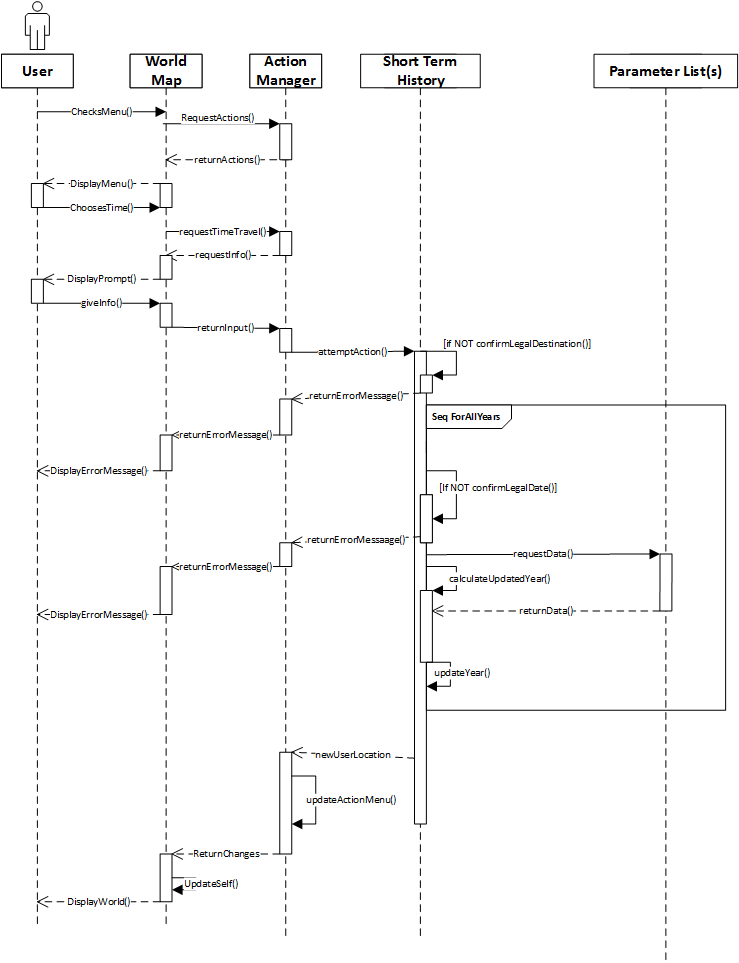
### Class Diagrams



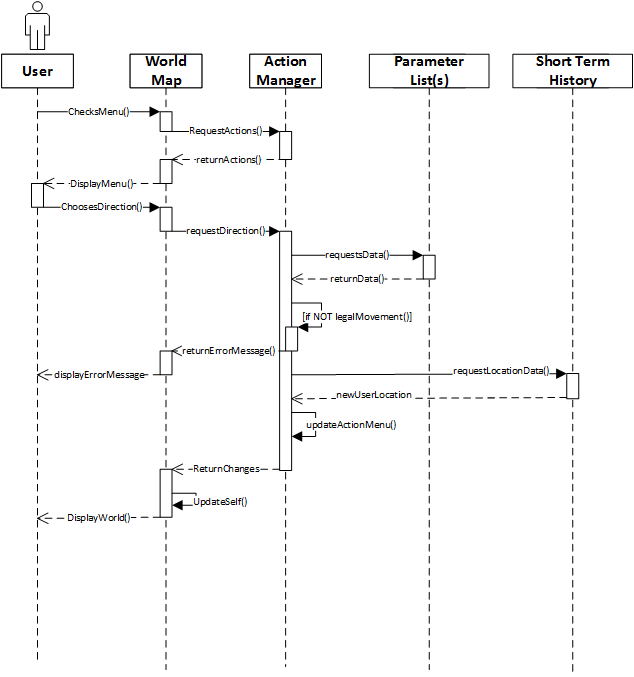


### Dynamic Model

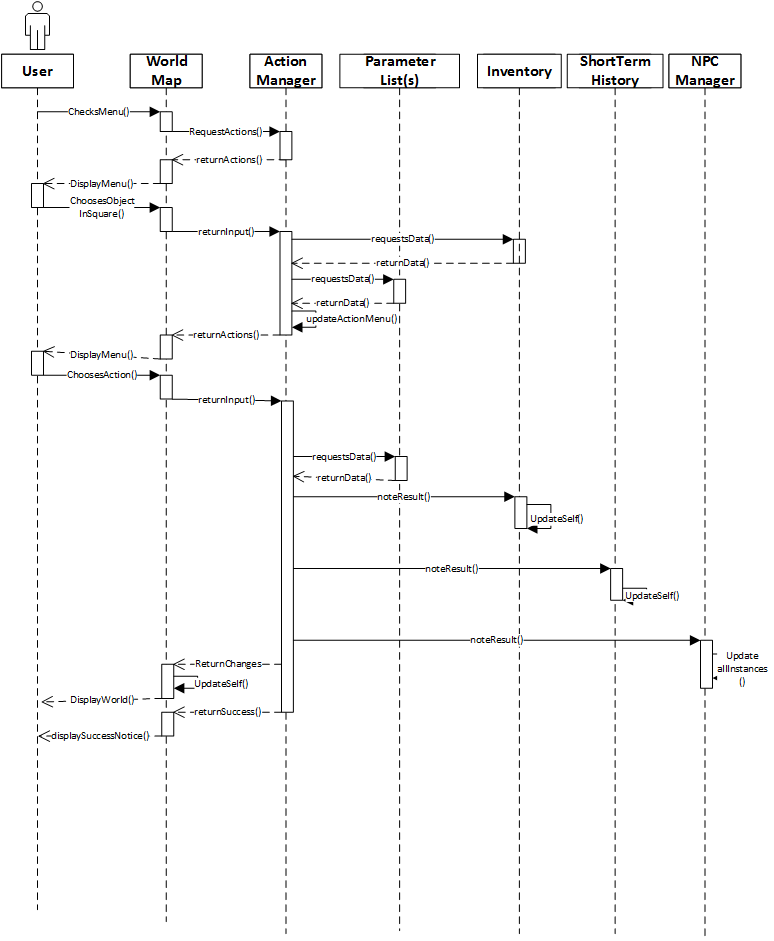
Move Through Time:



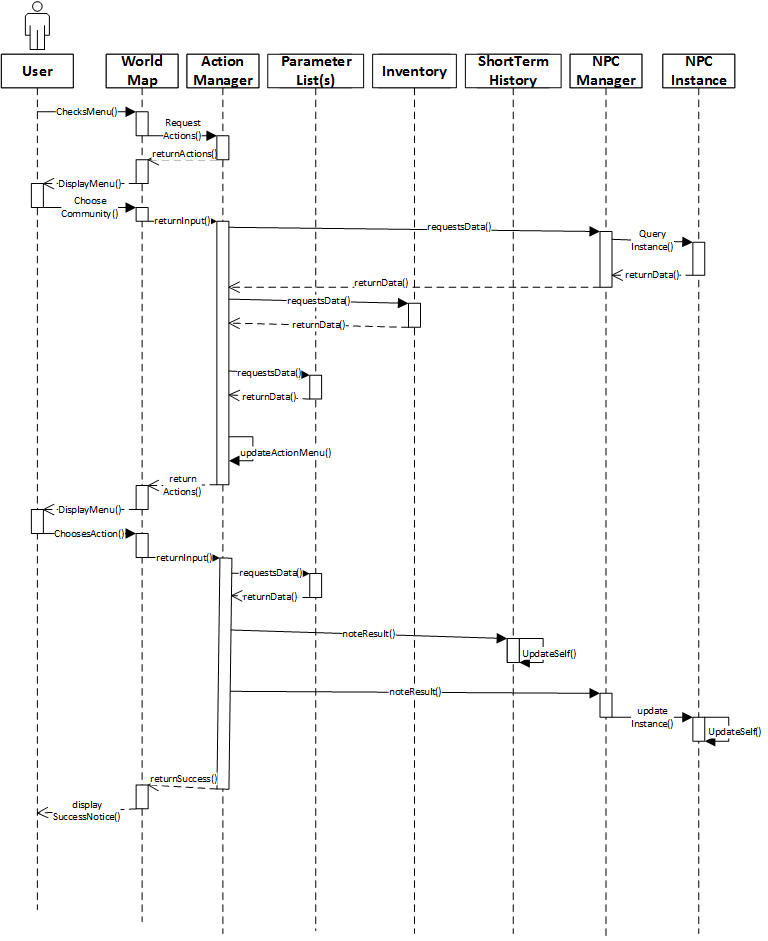
Over Land Movement:



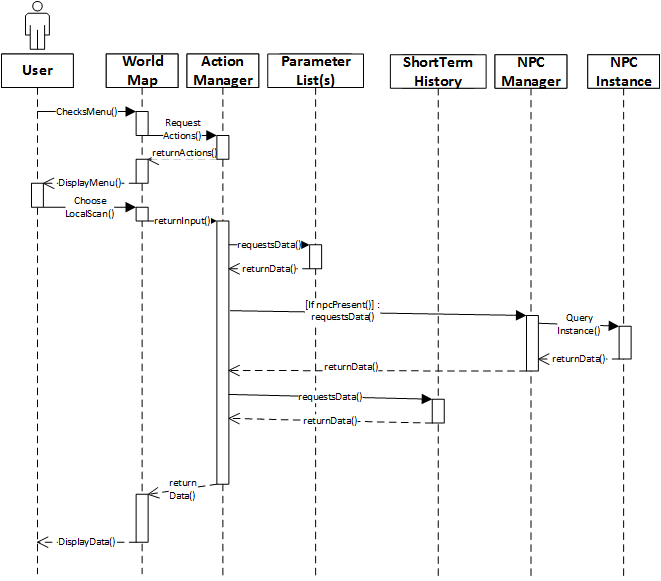
Interact with object in square:



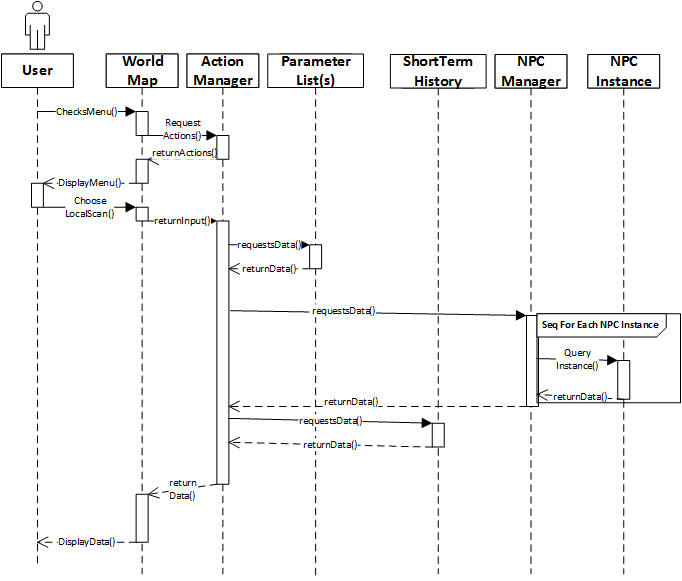
Interact with community:



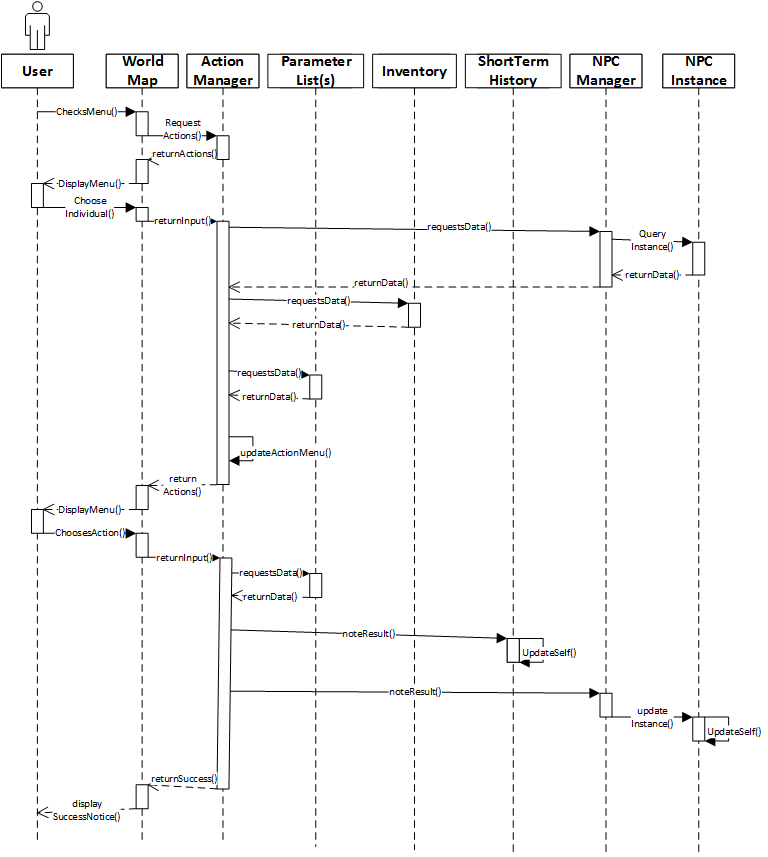
Local Scan:



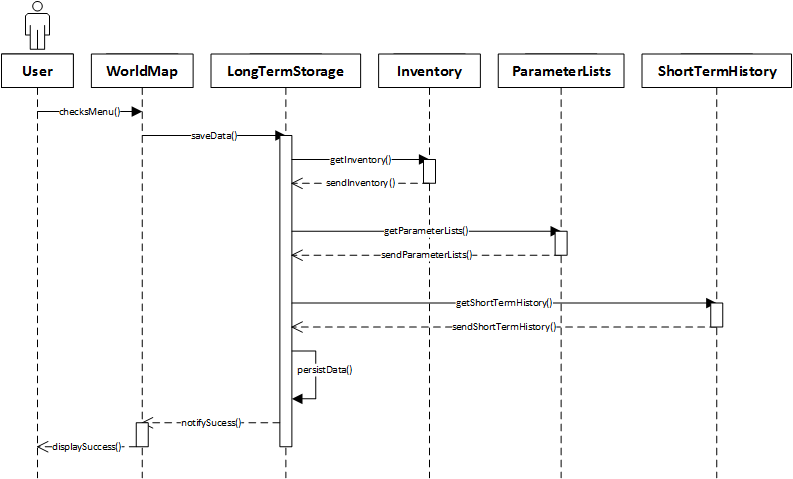
Global Scan:



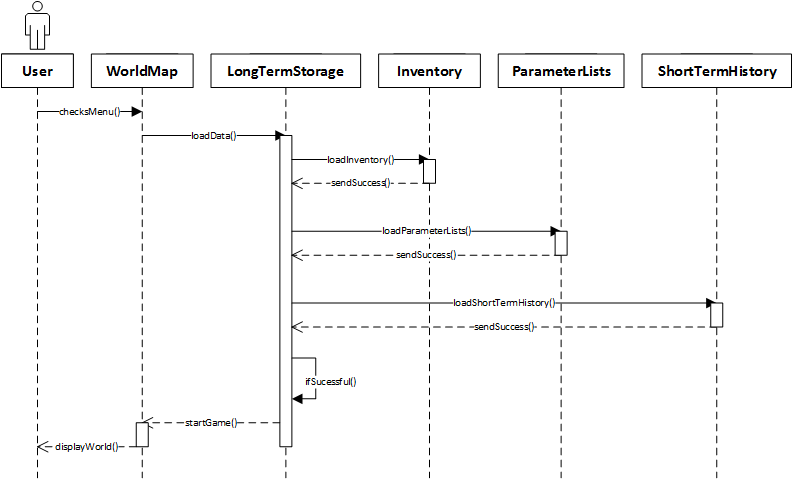
Interact with individual:



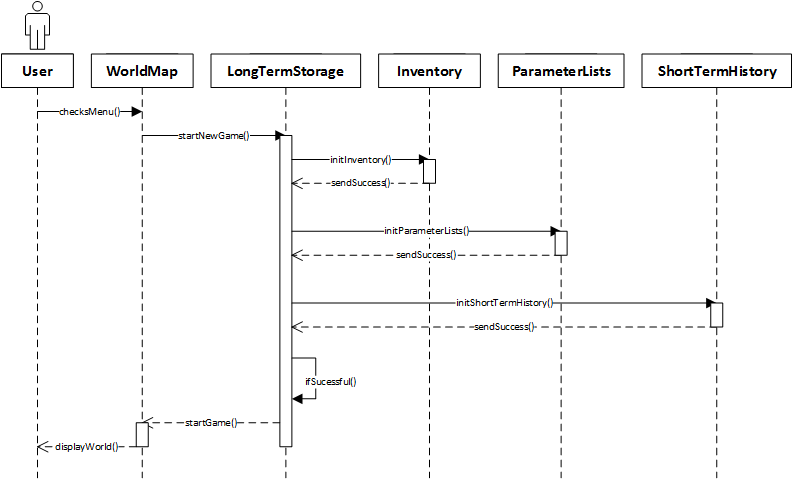
Save game:



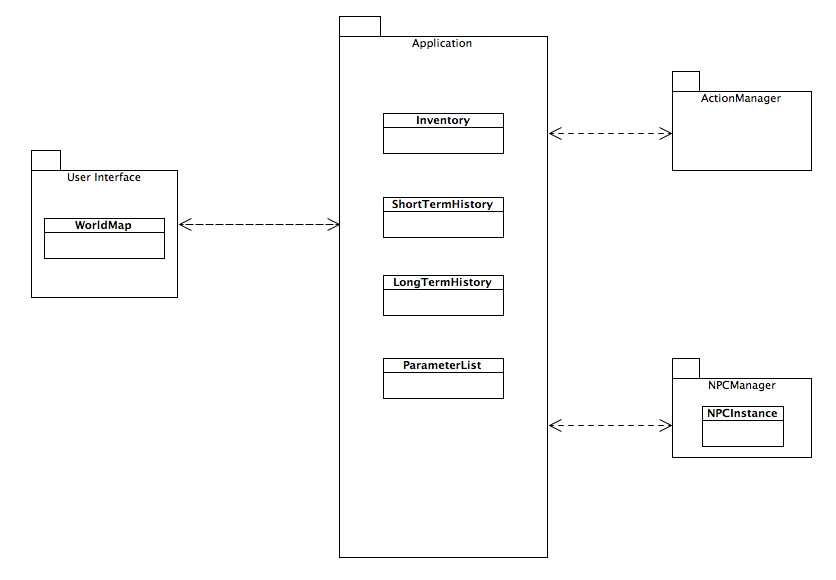
Load Game:



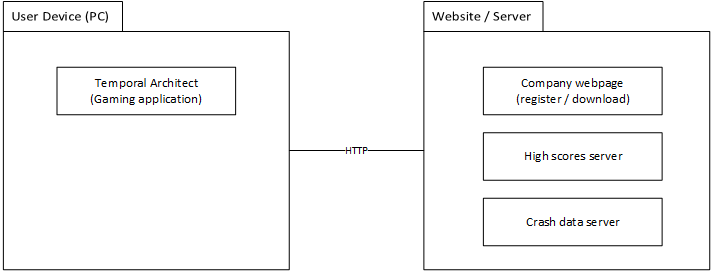
Start new game:



### Subsystem Decomposition



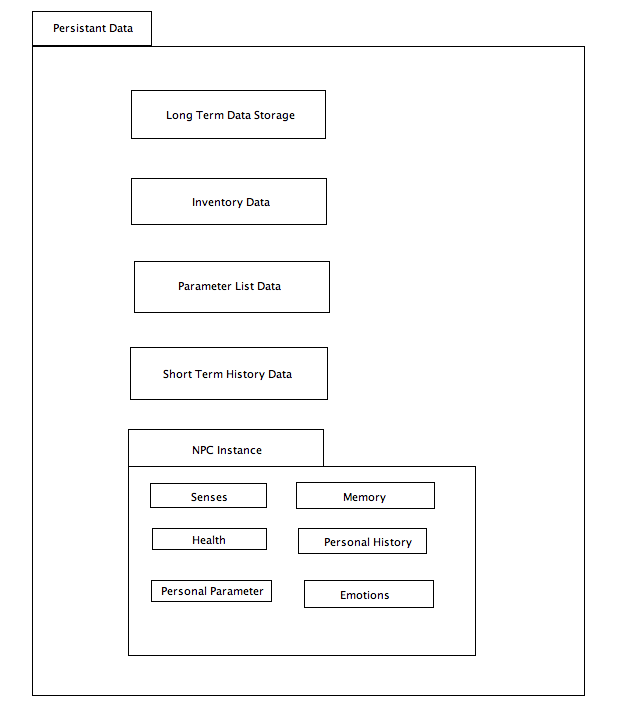
### Hardware / software mapping



### Data Dictionary

|  |  |
| --- | --- |
| Parameter | A rule that is applied to the in-game world or otherwise defines what the player can and cannot do. For example, “Action: Create Torch. Requires: Stick, Flint-and-Steel. Result: 1 Torch, -1 Stick” might be a parameter. Other rules like “Max Date: 2341” are possible as well. Parameters are all stored in the Parameter List(s) class, although each NPC instance stores some parameters specific to itself in its personalParameters class. |
| Year | A unit of measurement in the game, specifically the only way in which time is measured. The Short Term History List maintains a separate set of information for each year the player can travel too (and possibly years they can’t as well, depending on parameters) in a list of years (that are themselves lists or relevant data for the state of the world at that in-game unit of time). |
| Prompt | A little window that displays some information and possibly requests the User to type in some information (such as a year). |
| NPC Instance | NPC instance applies to both communities and individuals, and is a term that refers to a specific NPC instance manager and all of its related subclasses. A separate copy of all of these classes exists for every NPC in the game. Communities are treated as a single entity although some aspects of its internal functions (such as how many squares it can take up) are different. |
| Square | The only way space is measured in the game – each square is a square subsection of the world where various objects, geographic features, NPCs, etc can be. The world map is composed of many squares. |
| Avatar | The symbol/character that represents the User. What actions the User has available depends on the current in-game location of their Avatar. This game only supports one User playing at a time, thus there is only one Avatar in any given game session. |

### Persistent Data management



### Long Term Data Storage: Stores the state of a game session, including all of the other persistent data stored by any class. Its data is only accessed when loading a game, and is only changed when saving the game. In other words, this is a saved game file.

**Inventory Data:** Stores the contents of the player’s in-game inventory. Inventory data is only changed by the player adding/removing/interacting with the objects in their inventory (it is not effected by movement, traveling through time, etc).

**Parameter List Data:** The various parameters that define the rules of the in-game world. This includes how various items can be used, what dates can be traveled to, the initial starting state of the world, etc. The nature of the game can be greatly changed by using a different parameter list. The parameter list data is never altered for any given game world after being initialized for the first time (I.E., one must start a new game to use a different parameter list file [or load an old one that used a different list]).

**Short Term History Data:** Holds the state of the world for each in-game year in order to quickly load one date or another as time travel demands. This data is also used to calculate the effects of moving through time be using that state of one year to determine what happens the next year. The Short Term History Data’s stored data for whatever the current in-game year is changed whenever the player does anything that alters the state of the world.

**NPC Instance:** Various bits of information about each NPC needs to be kept. This information is stored in a number of subclasses for whom a copy exists for each NPC; the intent is for information that is only used by a single subclass class to be stored in that class. These classes are as follows:

**> Senses:** This class stores what senses the NPC has, the range of those senses, and how accurate its senses are. For example, a community may be able tell what is going on up to 10 squares away with perfect accuracy or it might only have a 50% chance to know what’s happening 3 squares away! Senses might also include being aware of pollution, knowing the mood of other NPC instances, etc. This data can be changed by the actions of the player, other NPC instances, or the NPC instance’s own actions.

**> Health:** This class stores the physical state of the NPC. This includes population size (individuals have size = 1), current moral, physical health (for communities this is an average for the population), resources (possessions and the like), etc. This data can be changed by the actions of the player, other NPC instances, or the NPC instance’s own actions.

**> Personal Parameters:** This class functions much like the parameter list(s) class, except it pertains to a specific NPC instance. An example of a parameter is whether the instance is an individual or a community. This data is not changed after the NPC class is initialized for the first time.

**> Memory:** This class keeps tracks of events the NPC finds important and the NPCs opinions on various other NPCs and objects (Perhaps it hates another community, or is obsessed with gold). Typically, more emotional memories hold greater influence. This data can be changed by the actions of the player, other NPC instances, or the NPC instance’s own actions.

**> Personal History:** This class functions much like the Short Term History class, except that it is specific to an NPC instance. It holds the state of the other subclasses for each year so that they can quickly be reloaded as the date changes. This data can be changed by the actions of the player, other NPC instances, or the NPC instance’s own actions.

**> Emotions:** This class holds the current emotional state of the NPC. Happiness, fear, anger, etc. This data can be changed by the actions of the player, other NPC instances, or the NPC instance’s own actions.

### Access control and security

The Temporal Architect will require the user to register a user account with the company/game website to gain access to the downloadable installation file and to have the opportunity and choice to download game extension files. This will also serve as a unique user name system to record and save user high scores with the high scores database service.

### Global software control

The gaming application will have a main game file that comes with the initial download.  If at any point the developers find a bug or error in the game file, a updated game file will be created and upon opening the gaming application the end user will be notified and asked if they would like to download the updated game file.

The website will contain a database of high scores from end users submitting their scores.  This database will be updated every time an end user submits a high score and will in turn update the high scores page on the website accordingly.

### Boundary conditions

The boundary conditions of the system specifies how the system is configured, started, shut down, and how potential errors are handled.

**Configuration:** Before allowing the user to download the game, they will be asked to create an account and pay a minimal fee for the initial gaming application.  The gaming application will be downloaded from the company website.

**Starting the game:** To start the gaming application the user will need to open the application.  Upon opening the application, the application will retrieve any saved games and allow the end user to pick from a list of saved game files or create a new game file.

**Exiting the game:**To exit the gaming application the user will click the “x” at the top of the application.  If the user has unsaved game data, the application will popup a window asking the user if they would like to save before exiting.

**Exception Handling:** If at any point during the end user playing the game application the application crashes, the end user will be asked if they would like to send a crash report to the company.  If the end user sends the crash report, a detailed report will be created and sent to the development team to help access what happened and what needs to be done to stop future crashes.

## Subsystem services

User Interface

The user interface will be what the end-user sees when playing the game and using the application. Different aspects of the user interface will include the main menu screen, the in-game menu screen, the inventory screen, and the main map screen (the visual game play screen).

Application Managers

The application managers will manage various aspects of the game. For example the “ActionManager” controls and manages various classes within the application that control general game play and reactions to user input. Also, the “NPCManager” controls what NPCs (Non-player-characters) arrive within the game during game play based on various results from the user input and actions.

Storage

Storage services will include all services that will operate on and with persistent data. For instance, when the user opts to save the game and quit his/her current gaming session, the storage services will save the required data to files as required to read back (load) into the application at a later date to effectively restore the game state to the last game state when saved.

High Scores

The high scores service will maintain scores for your gameplay. This will also send your high score to the high scores database associated with the website.

Crash Data

Crash data services will maintain all crash data for the user.  For instance, if the game crashes while the user is playing and the user selects to send the data, a crash data form will be generated and sent to the company crash data database.  All crash data can be accessed by the development team and deleted by them as well.

## User Interface

When user first launches the program the system displays an opening welcome screen with the options ‘Start New Game’, ‘Load Game’, and ‘Help’. Typing ‘Help’ and hitting enter, causes the system to display a summary of the game and how to both start it and play it.

Once the game has been started the system displays a numbered list of starting worlds – each world corresponds to a different data file in the program’s Worlds folder. The user decides to pick the first one (The first of the ‘standard’ worlds that came with the game) and so types ‘1’ and then hits enter. The program then reads the file, which tells it what other files to read while building the environment. The user’s screen shows a loading bar while the world is being built.

Once loading is complete, if this is a new game, the system displays a numbered list of potential victory conditions – supported victory conditions for this world were detailed in the parameter list for this world. Additional choices may be offered as well, depending on the parameter lists.

The system displays the user’s current square on a world map along with any nearby squares. A box in the top left corner displays the current year. The user has a numbered menu list on the side that details the various menus – 1 Movement, 2 Local Scan, 3 Global Scan, 4 Regular Actions, 5 Inventory, 6 Save, 7 Load, 8 Options, 9 Help, 10 Quit. Using the menu is either done by typing in numbers and navigating the menus or by using hotkeys/phrases that can be set under ‘options’. The hotkeys are initially set to being words/phrases such as typing “Local Scan” to instantly use Local Scan menu.

Images in each square give a very basic idea of what natural resources and civilizations exist in them, although more information on the current square can be displayed by typing ‘2’ (local scan) and then the number corresponding to the aspect of the world the user wishes to learn more about. The parameter files can give additional scan choices. Typing ‘3’ (global scan) causes a summary of the current state of the entire world instead, but otherwise is the same as local scan. Typing ‘4’ (regular acitons) brings up a list of all things within the square that the user can interact with. These interactions were defined in the data file, but the standard file handles interacting with key figures, natural phenomena such as forest and rivers, and a multitude of other options. Having certain objects in your inventory may change what interaction options you have access to, such as the user being able to plant a field if they have seeds. Typing ‘5’ in the main menu displays the user’s current inventory. Typing ‘6’ when the main menu is displayed saves the game. Typing ‘7’ shows a list of saved games to choose from.

## Object Design

### Object Design trade-offs

**Space vs. Response Time:** This gaming application should choose “Response Time” over “Space”.  Since the end user will be playing the game continuously the application should be quick to respond to the user’s input.

**Open Source Game Engines vs. Purchased Game Engines vs. In-House Build:** This gaming application should be built from the ground up by the development team instead of buying a third party’s gaming engine. 3rd party gaming engines (open source or purchased) may not have the AI logic and user input response reactions the Temporal Architect gaming application requires.

### Interface Documentation guidelines

The guidelines that must be followed for variables, methods, classes, Interfaces, and packages are as follows:

|  |  |  |
| --- | --- | --- |
| **Identifier Type** | **Rules for Naming** | **Examples** |
| Package | The package name must be a phrase or noun with each word after the first starting with an uppercase character. | aPackage |
| Class | The class name must be a phrase or noun with each word starting with an uppercase character. | class Inventory {} |
| Method | The method name must be a verb phrase with each word after the first starting with an uppercase character. | getMoves(void);  setMoves(); |
| Variable | The variable name must be a phrase or noun with each word after the first starting with an uppercase character. | int currentYear;  String playerName; |

### Packages

GUI:

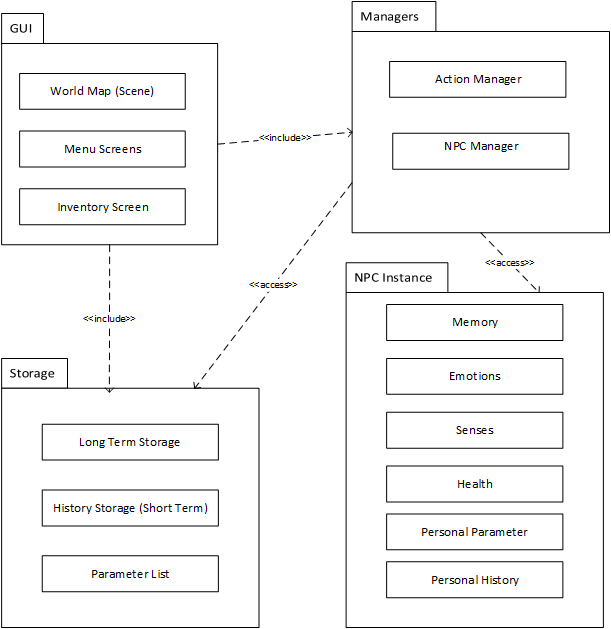
The GUI package contains all classes that assist in display content to the user on screen. For instance, the world map displays the actual scenery from within the game world to the users display. The menu screen classes will display menus as per the user request (the main menu screen for example).

Managers:

The managers package contains all classes that will effectively control the in-game events. Action manager updates all gaming lists and variables that store information on what happens next (for example a user action may alter the history storage list (short term).

Storage:

The storage package contains all lists, data structures, etc. that store in-game content and data that will be referenced and changed upon user input and actions.



### Class Interfaces

|  |  |
| --- | --- |
| **Class** | ActionManager |
| **Description** | The Action Manager manages the player’s actions. To be more specific, it keeps track of what the player is capable of doing at their current location in time and space (it also keeps track of where their avatar is). |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | returnActions | Returns the list of actions currently available to the user. | | DisplayPrompt | Displays a prompt to the user. | | UpdateActionMenu | Determines the what actions are now available to the user based on the given input. | | RequestData | Prompts another class for information on its contents. | | returnErrorMessage | Sends an error message to the World Map. | | returnSuccess | Sends a message noting an action has been successfully completed to the World Map. | | sendInput | Sends information based on the User’s actions to the action manager. | |

|  |  |
| --- | --- |
| **Class** | Inventory |
| **Description** | The Inventory keeps track of what in-game objects the User has in their possession. It returns requested data to the Action Manager. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | addItem | Adds an item to the inventory. | | removeItem | Removes an item from the inventory | | returnData | Returns a list of all object in the inventory. | |

# 

|  |  |
| --- | --- |
| **Class** | LongTermStorage |
| **Description** | The Long Term Storage stores all of the data held in the other classes as of the most recent save game action. This data can also be accessed in order to load the game. It interacts with all other classes, but only when saving or loading (not noted in the other class descriptions). |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | saveGame | Requests and then stores all of the stored data in all other classes. | | loadGame | Sends the proper input to all classes in order to initialized them to the saved state. | |

|  |  |
| --- | --- |
| **Class** | WorldMap |
| **Description** | The World Map manages the GUI interface. It interacts with the User and the Action Manager. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | DisplayWorldMap | Displays the world map on the computer screen. | | DisplayPrompt | Displays a prompt to the user. | | CalculateChanges | Determines the new state of the world map based on given input. | | RequestActions | Requests the action list from the action manager, displays the action list to the user. | | displayMenu | Displays a menu to the user than can be looked through so as to choose an action. | | sendInput | Sends information based on the User’s actions to the action manager. | | DisplayErrorMessage | Displays an error message based on given input. | |

|  |  |
| --- | --- |
| **Class** | ParameterLists |
| **Description** | The Parameter Lists hold all of the parameters for any given game world. This class holds data that is requested by the Action Manager, the inventory, and the Short Term history list. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | returnData | Returns a list containing the parameters corresponding to some specific input (or all parameters) | | setData | Initializes this class with the given input. | |

|  |  |
| --- | --- |
| **Class** | NPCManager |
| **Description** | The NPC Manager manages all of the NPC instances that exist in the game world. This includes accessing them, creating new NPCs, removing NPCs, etc. The NPC manager interacts with the Action Manager and the Short Term History list. It requests data from the parameter lists. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | AddNPC | Adds a newly created NPC to the list of NPCs. Can also be used to initialize a new NPC if given the input for it. | | ReturnNPC | Calculates and returns what the NPC has decided to do in a given situation | | ReturnData | Returns the Health and Opinion data of a specific NPC instance. Might return other data if noted by the world’s parameter lists. | | UpdateNPCs | Prompts each NPC present in the current in-game year for actions and then returns their results to them so that they may calculate their state in the next in-game year. | |

|  |  |
| --- | --- |
| **Class** | ShortTermHistory |
| **Description** | The Short Term History holds the data for the state of everything in the in-game world for each in-game year in a list of years. The Short Term History interacts with the Action Manager and requests data from the parameter lists. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | getShortTermHistory | Returns a list containing the stored information for the current year. | | setShortTermHistory | Alters the stored information for the current year based on the given input. | | CalculateYearChanges | Calculates and sets the stores information for a year based on the previous year. | |

|  |  |
| --- | --- |
| **Class** | NPCInstanceManager (A copy of this class exists for each NPC) |
| **Description** | The NPC Instance Manager manages the classes for whom separate copy exists for the same NPC instance the NPC Instance manager itself pertains to. It also contains the code for performing in-game actions. It sends and receives data/instructions from the Senses, Memory, Emotion, and Personal History classes. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | ReturnHealthData | Returns the various values stored in the Health Class. | | ReturnOpinonData | Returns the NPC’s opinion on something. What that thing is depends on the method input. | | InputData | Receives input that is then used to determine what inputs to send to other classes. | | SpawnNewIndividual | Creates a new NPC instance, one that is an individual. | | SpawnNewCommunity | Creates a new NPC instance, one that is a community. | | Expand | [Only usable by a community] – Increases the number of squares the NPC occupies and sends this information to the NPC manager. | | Attack | Attacks another NPC instance. The data regarding the type and effectiveness of the attack is sent to the NPC manager which sends the being attacked input to the other instance. | | AlterGeography | Alters geography in the same square as the NPC Instance. | | SpecialAction | What this method does depends on the Parameter and Personal Parameter lists; more than one special action might exist. | |

|  |  |
| --- | --- |
| **Class** | Senses (A copy of this class exists for each NPC) |
| **Description** | The Senses holds the information pertaining to an NPC’s senses. In addition, this class calculates what an NPC senses and sends that data to an NPC instance’s Memory, Health, Personal History, and Emotions class. This class receives prompts for information from the NPC Instance manager. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | FindData | Returns a request for information that is then used to calculate what the NPC senses. | | AlterSensoryRange | Alters/sets the range of a sense. | | AddNewSense | Adds a new sense. | | RemoveSense | Removes a sense. | | RetrieveData | Returns the data that the NPC has most recently acquired via its senses. | |

|  |  |
| --- | --- |
| **Class** | Memory (A copy of this class exists for each NPC) |
| **Description** | The Memory keeps tracks of events the NPC finds important and the NPCs opinions on various other NPCs and objects (Perhaps it hates another community, or is obsessed with gold). This class interacts with the Emotions, receives data/instructions from the NPC Manager and Senses classes, and sends data to the Personal History class. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | RecallReleventMemory | Returns whichever memory is deemed most relevant to the given input. | | StoreMemory | Creates and stores a memory based on the given input. | | RecallOpinion | Returns what the NPC’s previous opinion of another NPC, an object, or the User is. | |

|  |  |
| --- | --- |
| **Class** | Health (A copy of this class exists for each NPC) |
| **Description** | The Health stores the physical state of the NPC. This includes population size (individuals have size = 1), current moral, physical health (for communities this is an average for the population), resources (possessions and the like), etc. This class interacts with the NPC manager class and sends information to the Memory, Emotions, and Personal History classes. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | CalculateModifications | Modifies the internal data of the class based on given input. If class has not been initialized, initializes the class. | | ReturnPhysicalData | Returns the values of each of this class’ variables. | | returnChanges | Returns the values of the changes caused by the last CalculateModifications call. I.E., the difference between the old and new values. | |

|  |  |
| --- | --- |
| **Class** | PersonalHistory (A copy of this class exists for each NPC) |
| **Description** | The Personal History stores the state of the other classes specific to the same NPC for each in-game year. It receives information from the Memory, Emotions, Senses, and Health classes and can be used to initialize the these classes when the date changes to a previous date. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | addData | Adds data to the current year. All additions overwrite any preexisting data for the given statistic, so this can also be used to initialize the class. | | returnData | Returns the data for the current year. | |

|  |  |
| --- | --- |
| **Class** | PersonalParameters (A copy of this class exists for each NPC) |
| **Description** | The Personal Parameters store the parameters specific to the NPC instance for whom any given copy exists for. This includes whether an NPC is a community or an individual. The Personal Parameters are initialized by the NPC Instance Manager but otherwise is read-only. It sends data to the Senses, Health, Emotions, Memory, and NPC Instance Manager classes. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | returnParameters | Returns a list containing the parameters corresponding to some specific input (or all parameters) | | setParameters | Initializes this class with the given input. | |

|  |  |
| --- | --- |
| **Class** | Emotions (A copy of this class exists for each NPC) |
| **Description** | The Emotions handle the emotional state of a specific NPC instance and determines what decisions the NPC makes. It both sends are receives data from the NPC Instance Manager and Memory Class, is sent data from the Health, Senses, and NPC instance Manager class, and sends data to the Personal History class. |
| **Methods** | |  |  | | --- | --- | | **Name** | **Description** | | ReturnEmotionalStatistics | Returns a list containing the value of every emotion | | ReturnDecision | Calculates and returns what the NPC has decided to do in a given situation | | returnOpinon | Calculates and returns what the NPC’s opinion of another NPC, an object, or the User is. | | SetEmotionalState | Initializes the NPCs emotional state based on given input. | |

# Test Plans

## Features to be tested / not to be tested

* Website Registration
* Start/Load/Save/Quit Game
* Directional Movement
* Manipulate Inventory/Craft Item
* Travel through Time
* Interact with NPCs
* Object Interaction

## Pass/Fail Criteria

Unit Testing:

* All units must pass testing with a 99% pass rate.
* All units must be fail safe.

Integration Testing:

* All integration tests must pass with a 95% pass rate.
* All bugs that crash or create a lengthy lag to the game must be fixed before release.

Systems Testing:

* All functional requirements must pass with a 99% pass rate prior to release.
* All acceptance testing must receive 100% pass rate before full release.

Pilot Testing:

* All bugs and errors found during alpha testing must be fixed before beta testing.
* All bugs and errors found during beta testing must be fixed before full release.

## Approach

Unit Testing – Used to test for faults in all of the smallest units from the use case model of the program.

Integration Testing – The approach followed is “Modified Sandwich Testing”. This strategy will test three layers before integrating them together to perform more tests.

Systems Testing – Used to test if the completed system complies with the functional and non-functional requirements.

## Suspension and resumption

During testing, if one unit/integrated test passes with less than 10% pass rate, the issue must be fixed before continuing.

## Testing materials ( hardware / software requirements )

Hardware Requirements:

1. Operating System: Windows/IOS/Linux
2. Memory: 2GB minimum (3GB for windows 7 or 8)
3. Processor: Intel Core 2 duo 2.4gh or similar
4. Graphics: OpenGL 2.0 compatible
5. Hard Drive: 2GB available space

Software Requirements:

1. Operating System: Windows/OSX/Linux
2. A modern browser that allows downloading from the internet

## Test cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case # | Test Case Name | Execution Steps | Expected Result |
| 1 | Website Registration | Preconditions:   * Access to Internet   Steps:   1. Visit Company Website 2. Click “Register” 3. Enter your information | User has a valid username/password. |
| 2 | Download Game | Preconditions:   * Valid account * Form of Payment   Steps:   1. Go to company website 2. Login 3. Navigate to Download page 4. Click “Purchase” 5. Enter payment information | The payment is accepted and the game downloads. |
| 3 | Install the Game | Precondition/s:   * User has downloaded game.   Steps:   1. Click on the installation package. 2. Accept the terms of agreement. 3. Click “Install” | The game is installed successfully and ready to play. |
| 4 | Launch Game | Preconditions:   * The game must be installed.   Steps:   1. Start the executable or double-click the shortcut icon. | The application opens. |
| 5 | Start Game | Preconditions:   * The application is open and running and the main menu interface is displayed.   Steps:   1. Navigate the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘New Game’) or press a previously set hotkey and then the number corresponding to ‘New Game’. 2. Choose a parameter files from a list of those currently in the game’s parameter file folder. | A new game begins. |
| 6 | Save Game | Preconditions:   * The applications main menu interface has been invoked.   Steps:   1. Navigate the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Save Game’) or press a previously set hotkey and then the number corresponding to ‘Save Game’. 2. Enter a desired save game name. | The application creates a save game file and the required save game information is added to the file. |
| 7 | Load Game | Preconditions:   * The applications main menu interface has been invoked. * Prior games have been saved by the user.   Steps:   1. Navigate the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Load Game’) or press a previously set hotkey and then the number corresponding to ‘Save Game’. 2. Select the desired saved game from the available choices in the load save game list of options. | The application will read the desired save game file and load all save game parameters and the appropriate parameter file for that particular game. |
| 8 | Quit Game | Preconditions:   * The application is running. * The user has invoked the main menu screen.   Steps:   1. Navigate the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Quit Game’) or press a previously set hotkey and then the number corresponding to ‘Quit Game’. | The applications exists without error.  The application should also query the user if game play has advanced beyond the prior save (if a prior save exists). |
| 9 | Invoke Main Menu | Preconditions:   * The application is running.   Steps:   1. During game play the user presses the TAB key at any time or right clicks any part of the screen. | The main menu is displayed as a UI overlay. |
| 10 | (In Game)  Travel in a direction. | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘1’ for ‘movement’ and then one of the following:‘1’ for North, ‘2’ for South, ‘3’ for East, or ‘4’ for West) or presses a previously set hotkey. | The User’s avatar has traveled one square in the desired direction. |
| 11 | (In Game)  Travel Through Time | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘1’ for ‘movement’ and then ‘7’ for ‘Time Travel’) or presses a previously set hotkey. 2. User types in some date when so prompted by the program. | Either the User’s avatar is has traveled to the desired time or an appropriate error message has been displayed (if going to a restricted year/ before year 0/etc). |
| 12 | (In Game)  Acquire Object | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:  1. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the object. | The User has acquired the selected object (assuming it is possible to acquire that object) |
| 15 | (In Game)  Use Object | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘5’ for ‘inventory’ to bring up a list of all objects in the User’s inventory along with their associated number or presses a pre-set hotkey to do the same. The User then presses the number corresponding to the Object they would like to use. 2. The User, now given a list of potential uses for their object, chooses one such use via the menu. | The object has been used in some way, as determined by the User. |
| 16 | (In Game)  Interact with object. | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to the object. 2. The User, given a list of potential actions, inputs the number of the action they would like to take. | The User’s avatar has interacted with an object in the manner they chose. |
| 17 | (In Game)  Interact with Geography | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding a geographic feature. 2. The User, given a list of potential actions, inputs the number of the action they would like to take. | The User’s avatar has interacted with an Geographic Feature in the manner they chose. |
| 18 | (In Game)  Interact With NPC | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘4’ for ‘Regular Actions’ to bring up a list of all interactable objects/communities/geography/etc and then pressing the number corresponding to the object) or presses a previously set hotkey to bring up the list of all interactable objects/communites/etc and then presses a number corresponding to an NPC. 2. The User, given a list of potential actions, inputs the number of the action they would like to take. | The User’s avatar has interacted with a NPC in the manner they chose. |
| 19 | (In Game)  Use Local Scan | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘2’ for ‘Local Scan’) to bring up a list of all information groups. 2. User presses the number corresponding to the desired group. | The game displays the desired information pertaining to the current square the User’s avatar is in. |
| 20 | (In Game)  Use Global Scan | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘3’ for ‘Local Scan’) to bring up a list of all information groups.   2. User presses the number corresponding to the desired group. | The game displays the desired information pertaining to the entire world. |
| 21 | (In Game)  Confirm Victory | Preconditions:   * The application is running or has just been started. * The menu is being displayed. * The User has met the conditions needed to win the game.   Steps:   1. User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Confirm Victory’) or presses a previously set hotkey and then the number corresponding to ‘Confirm Victory’. | The User is displayed a prompt congratulating them. Also, the User’s high score has been calculated. |
| 22 | (In Game)  Set Hot Keys | Preconditions:   * The application is running or has just been started. * The menu is being displayed.   Steps:   1. User either navigates the menu (pressing ‘3’ for ‘Local Scan’) to bring up a list of all information groups. 2. User either navigates the menu (pressing ‘6’ for ‘Options’ and then the number corresponding to ‘Set Hot Keys’) or presses a previously set hotkey and then the number corresponding to ‘Set Hot Key’. 3. The User, having been displayed a list of current hot key bindings, presses a number corresponding with a hot key and then a new key that becomes bound to the previous hot key. 4. Step 3 can be repeated any number of times until the User presses the number corresponding with ‘return to game’. | The in-game Hot keys have been changed to the chosen values. |

## Testing schedule

|  |  |  |  |
| --- | --- | --- | --- |
| **Week(s)** | **Test Type** | **Date(s)** | **Test Case(s)** |
| 4th – 12th week | Unit testing and Integration testing | 02/02/2015 – 04/17/2015 | Test case #5 through Test case #22 |
| 12th – 14th week | System Testing | 04/20/2015 – 04/25/2015 | Test case #1 through Test case #4 |
| 14th week | Inspection Testing | 04/27/2015 – 05/01/2015 | n/a |

# Project Issues

## Open Issues

We have still not decided what the actual format of the parameter files are.

We have not decided how the scoring system should work for the game. Because each parameter file will have a different information and objectives, we need to do more research before deciding exactly how to score players.

## Off-the-Shelf Solutions

### Ready-Made Products

* 1. Existing game engines may be used in place of creating the engine from scratch if the existing engine will satisfy all requirements.
  2. Server Hosting Sites to host the company webpage

### Reusable Components

Since this is our first product, there are no previous components made that can be used for our product.

### Products That Can Be Copied

Existing two-dimensional “point-and-click” games may be referenced to assist in engine creation.

## New Problems

### Effects on the Current Environment

The product shall not affect the current environment.

### Effects on the Installed Systems

Temporal Architect is a new standalone desktop application. It should not interact with any installed systems and should not interact with any pre-existing systems.

### Potential User Problems

If the user has never played a “point-and-click” long-term strategy style game in the past, the learning curve (however small) may pose a problem. The user may not get the full experience of the gaming application until he/she understands the game mechanics.

### Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

The users system may not be able to handle extremely large numbers of NPC instances. This depends greatly on the exact implementation, and might result in their needing to be some sort of limit.

### Follow-Up Problems

If the game becomes extremely popular, we might exceed the amount of bandwidth aloud on the hosting site we choose.

## Tasks

### Project Planning

|  |  |  |
| --- | --- | --- |
|  | **Time** | **Developers** |
| Requirement #1 | 250 hours | 4 |
| Requirement #2 | 200 hours | 3 |
| Requirement #3 | 300 hours | 5 |
| Requirement #4 | 125 hours | 2 |
| Requirement #5 | 200 hours | 3 |
| Requirement #6 | 200 hours | 3 |
| Requirement #7 | 300 hours | 5 |
| Requirement #8 | 350 hours | 6 |

### Planning of the Development Phases

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Operational Date** | **Operating Environment** | **Functional Requirements** |
| Version 0.1 | 6 months in | In-house | 2, 3, 5, 6, 7 |
| Version 0.9 (beta) | 10 months in | In-house / On-site | All |
| Version 1.0 (release) | 12 months in | On-Site | All |

## Migration to the New Product

### Requirements for Migration to the New Product

As this is the first release of this software application package, no requirements for migration from an older product have been written.

### Data That Has to Be Modified or Translated for the New System

As this is the first release of this software application package, no data need to be modified.

## Risks

|  |  |  |
| --- | --- | --- |
| **Description** | **Probability** | **Cost** |
| Customer pushes release ahead of schedule by 1 month | 0.75 | Developers will have to be paid overtime increasing labor cost. |
| Customer pushes release ahead of schedule by 1 month | 0.70 | Developers will have to be paid overtime increasing labor cost. |
| Customer pushes release ahead of schedule by 1 month | 0.25 | Developers will have to be paid overtime increasing labor cost. |
| A requirement will run past the time allocated to it for development. | 0.90 | Teams will have to be restructured which will cost development time. |
| Team developers / managers resign or otherwise cannot fulfil their duty. | 0.75 | Teams will have to be restructured which will cost development time. New labor may need to be hired depended on severity of issue. |
| Product does not pass customer’s acceptance test. | 0.25 | The product will need to be partially re-developed to meet specifications. |
| Customer will add new features mid-development. | 0.80 | Finished portions of the product may need to be re-developed. Teams may need to be restructured to address the new demand. |

## Costs

Plan A: We just hire a bunch of college students (undergraduate and graduate) to implement all facets of the application to give them a sense that unpaid internships actually mean something. We will probably need to pay a graphic artist for images, which will probably run us a few hundred dollars (2D, simplistic design).

Plan B: Same as plan A, except we actually pay the students. A team of 3-5 should be able to implement most of the features a few months, although implementing the AI to our specifications could take longer (half a year or more). If we pay similarly to RA stipends (minus waivers) that will run us about 10,000 a month for a team of 5. 100k (10 Months) should more than cover such expenses over the course of the project

## Waiting Room

1. Multiplayer capabilities. The ability for the user to join a network and deploy cooperative play with other players shall be considered for future releases.
2. Being able to manipulate multiple worlds in the same solar system.
3. Manipulating the Weather. This is likely just a parameter file addition, but it would be a huge one.
4. Burrowing/Flying.

## Ideas for Solutions

Recommended solutions and all meetings regarding adjustments, question sessions, updates, etc. shall be recording in a professional meeting notes environment such as Jira or Confluence.

## Project Retrospective

> Microsoft Word is somewhat of a pain to make heavily templated documents like this was in.

> Google Drive documents are very useful for working on a shared file, although copy/pasting from them into this word document was a pain.

> Google Hangouts, also very useful.

> Having a good design document (like this one) to compare with is very helpful when creating your own! It is a shame the one we got stuck with was awful, a fact that didn’t present itself until we got a group from last semester to show us their document (okay, it was clearly a lazy pile before that, but we didn’t know just HOW BAD it was.)

> Sure was a hell of a lot of fun. Yep.

# Glossary

**Neighborhood Informant:** A horrible project that was completed under the same company.

# References / Bibliography

This section describes the documents and other sources from which information was gathered. This sample bibliography was generated using the “Insert Citation” and “Bibliography” buttons in the “Citations & Bibliography” section under the “References” tab of MS Word. Creating new citations will not update this list unless you click on it and select “Update Field”. You may need to reset the style for this paragraph to “normal” after updating.

|  |  |
| --- | --- |
| [1] | Robertson and Robertson, Mastering the Requirements Process. |
| [2] | A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, Ninth ed., Wiley, 2013. |
| [3] | J. Bell, "Underwater Archaeological Survey Report Template: A Sample Document for Generating Consistent Professional Reports," Underwater Archaeological Society of Chicago, Chicago, 2012. |
| [4] | M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004. |

# Index

This section provides an index to the report. The sample below was generated using the “Mark Entry” and “Insert Index” items from the “Index” section on the “References” tab, and can be automatically updated by right clicking on the table below and selecting “Update Field”. To remove marked entries from the document, toggle the display of hidden paragraph marks ( the paragraph button on the “Home” tab ), and remove the tags shown with XE in { curly braces. }

Design 61, 63

Requirements 35, 51, 58

Test 64, 65