[THE ORB]

[Team IT'S A TRAP]

[AKBAR]

Bueckert, Ricky
Conrad, Lorenzo
Richards, Mathew

Mar. 15, 2019
Version 2.0

Table of Contents

Introduction

The project is a text based adventure game called [THE ORB]. The Orb is a game in which the object is to collect the required password, return to the location of The Orb, and input the password into The Orb. The pathway through the game is semi-linear, but around the midpoint you can obtain a teleporter that allows you to travel to different game locations in a non-linear pathway. Without the teleporter the progression through the game is linear in manner. Actions in the game include: interacting with non-playable characters, picking up or using items, solving puzzles, combat, movement, and even solving a timed event puzzle. This report includes details about how the team will be organized for the development of the game, as well as how the team will respond to potential problems that could appear during the development of the game.

Project Management

The team's success in this project will be largely reliant on implementing strong project management techniques. Clear delegation of different tasks and responsibilities will allow group cohesion and synergy. Paired with strong communication and a clear directive for the group, we anticipate to make [THE ORB] a reality by the implementation deadline.

Team Organization

Mathew Richards
- Team Lead, Software Developer, Software Tester

Lorenzo Conrad
- Quality Assurance Lead, Documentation Lead, Software Developer, Software Tester

Ricky Bueckert
- Design Lead, Software Developer, Software Tester

Risk Management

The problems/risks include: the planned project was too large, team underestimated how long parts of the project would take, major changes are needed during implementation, addition or loss of a team member, unproductive team members, team members lacking the technical background, major life events, inexperience with new tools, learning curve for tools is steeper than expected, and tools do not work together in an integrated way.

1) Requirements/Design/Estimation

- Risk 1: The planned project was too large
    - If the project we planned is too large we would prioritize the core of the game and make the game more linear to make the coding simpler and less time consuming. The core of the game being prioritized simply means that the important features and classes are done first reducing the number of items that are required to complete the final objective. When deciding what to drop to reduce the project size we would drop things that had not been started, meaning if we already implemented part of a secondary objective we would finish it and remove another secondary objective from the game. The reason for this is so that we do not waste time and effort already put into the project.
- Risk 2: Team underestimated how long parts of the project would take
    - If the team underestimated how long different parts would take then similar actions would be taken as if the project was planned too large. The team would re-assign resources to get the core of the game finished. We would re-assign people working on secondary systems to help finish the core classes in order to get back on schedule. We may need to drop secondary objectives like a timed event, which at present none of the group members know how to implement. However with deadlines set during team meetings as well as constantly pushing to the repository to see the progress of the separate parts we hope to catch this problem before it becomes a problem. If noticed in time we can adjust the timeline to better suit the project goals and deadline.
- Risk 3: Major changes are needed during implementation
    - If major changes are needed the first thing we would do is hold a team meeting to get consensus of the changes and delegate actions in the new plan. Then we would reallocate resources in order to prioritize the changes that need to be implemented. Meanwhile, less critical systems would take a back burner. We would also update our diagrams to match the change being made so that there is no confusion when implementing the changes and time is not wasted on coding things we are no longer implementing.

2) People

- Risk 4: Addition or loss of a team member
    - If we lost or added a team member, we would first call a team meeting with the new team to discuss the future of the project. In the case of a lost team member we may have to downsize the game to something manageable with fewer people. In the case of an addition to the team we would bring that person up to speed as fast as possible and delegate the current workload taking into consideration the skill set of the new team member.
- Risk 5: Unproductive team members
    - If we have unproductive team members, our first consideration would be to check if the goals we have set are achievable within the given time. Then we would talk to them about their responsibilities and ask them to increase their productivity. However, if team

members are not meeting the goals set out during the team meetings continuously, we would report this to the professor and delegate the tasks to the remaining team members so that the credit is given to the team members who are doing the work. If the project becomes to much for the productive team members, we would consider dropping secondary objectives from the game and simplifying the game.

- Risk 6: Team members lacking the technical background
    - All team members are fluent at an intermediate level in C++, which will be used to program the game. If there are difficult sections each member will try to research for themselves how to overcome the obstacle. Then, if the problems remain, we would consult each other, then an instructor. If a group member is unable to program a section we would re-assign their tasks and give them easier tasks to complete, also notify the professor of the situation in order to give credit to the team members who deserve it.
- Risk 7: Major life events
    - If a team member encounters major life events causing them to be less productive, we would try to accommodate the best we could, but we would notify the professor so the credit for the work that is done is given to the people who did it. Then we would delegate the workload to the remaining team members however if it is too much for two people to handle, we would simplify the game a bit and remove some of the secondary objectives/systems and focus our effort of the core of the game.

3) Learning & Tools

- Risk 8: Inexperience with new tools
    - In the case that the team is inexperienced with a new tool, we would have a team meeting to discuss how the tools are supposed to work (a kind of workshop to figure out together where and how to use the tool). If, however we cannot figure out the tool is supposed to be used as a team then we would talk to the professor to see if there are better tools to use that are simpler or easier to implement while still do the same thing.
- Risk 9: Learning Curve for tools is steeper than expected
    - If the learning curve for the new tool is steeper than we initially thought, we would see if there tools that are easier to implement and understand without losing functionality. The team would try to stick to the tools taught in class as much as possible so that we can ask the professor/TA for help with the tools. As well as being able to ask for help when using tools that are not taught in class, the responsibility lies solely on the team to make sure the tools works properly in the first place.
- Risk 10: Tools do not work together in an integrated way
    - If the tools that we are using are not working together in an integrated way, the first thing we would do is find out if our implementation of the tools is correct. If the implementations were correct then we would try find out if there are tools being used that could be replaced with another tool without losing functionality or efficiency of the project.

Development Process

This section outlines how team members in Team Akbar will establish good practices in code review, communication, change management, and software design. Establishing these practices prior to the implementation phase will allow for a smoother and more streamlined programming experience.

Code Review Process

For the code review process, each group member will work separately on their respective branch when changes are made and solidified then the group member will request a merge request with the master. This is so that the Quality Assurance Lead can go through the code to double check that is will not break the pipeline. This ensures that code will have been reviewed at least twice before added to master; once by the primary author, and then again by the QA lead.

Communication Tools

The team will be using WhatsApp to communicate with each other because it allows us to call, message and also send pictures. The reason pictures are helpful in this case is because if anyone is having trouble with a certain line of code a fresh set of eyes is always good to help and find the problem. If WhatsApp becomes too unreliable because of the University of Lethbridge's wireless network, then we will switch to Slack.

Change Management

Bug Reports will be delegated by the team lead based on the priorities that have been discussed by the group (with core programs/ code obviously have priority over sub programs) and giving it to the person/people who wrote that section of code that is causing the bug. When the bug is fixed the Quality Assurance Lead will double check if the bug is fixed and report thus clearing the bug report.

Software Design

The software design is shown in detailed UML diagrams in the appendices.

Appendices

Appendix A: Figures and Tables

Class Diagrams

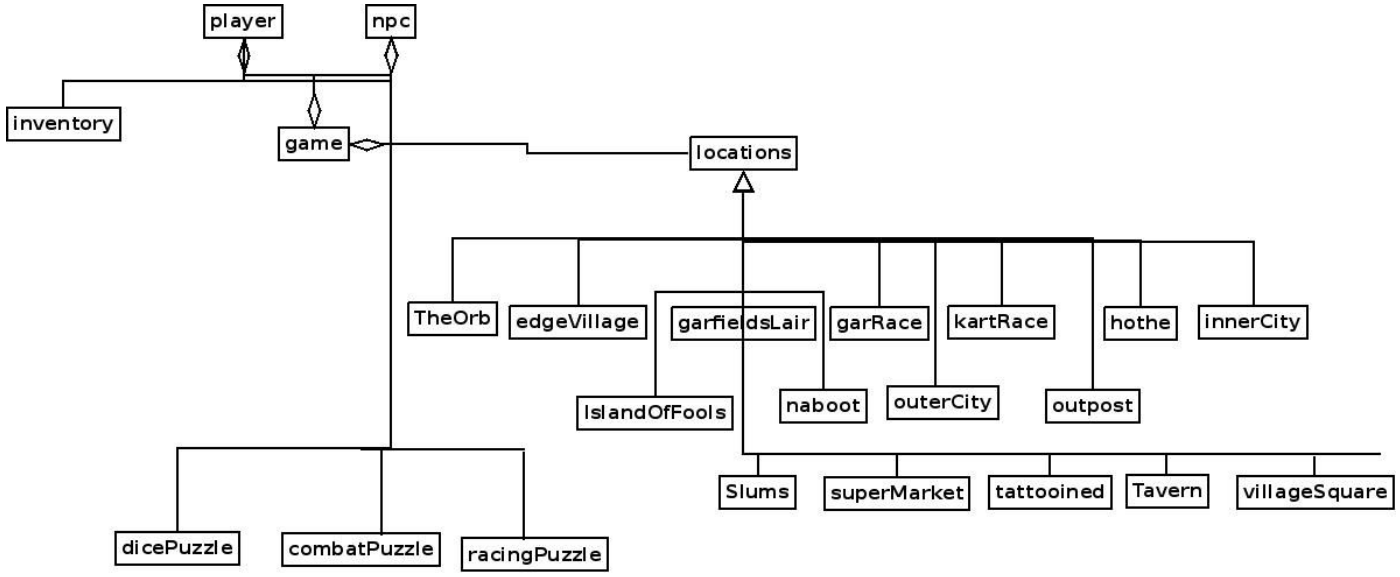

Figure 1 - Overview of classes (does not include exception classes)

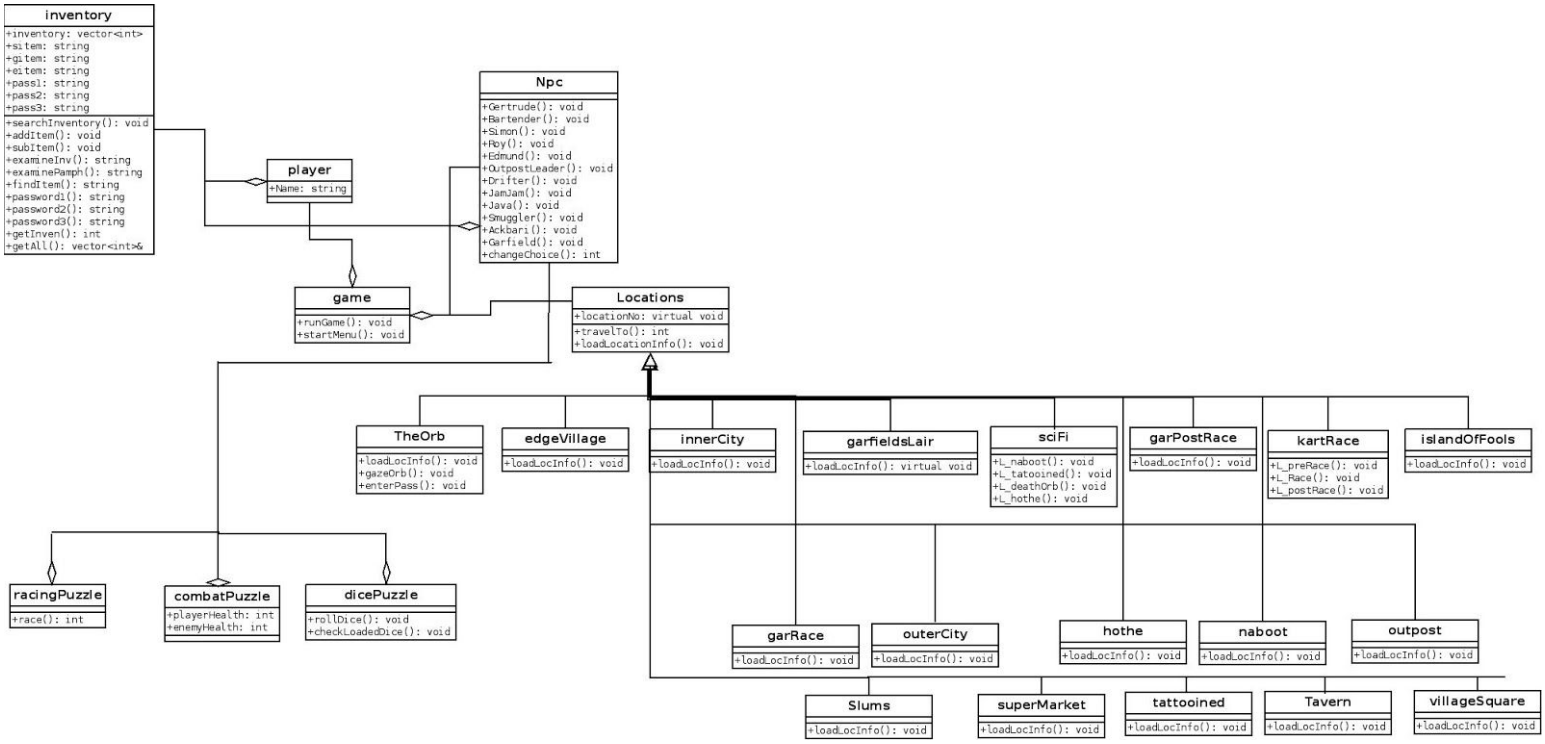

Figure 2 - Implementation of character classes and relationships, locations, and game state control.

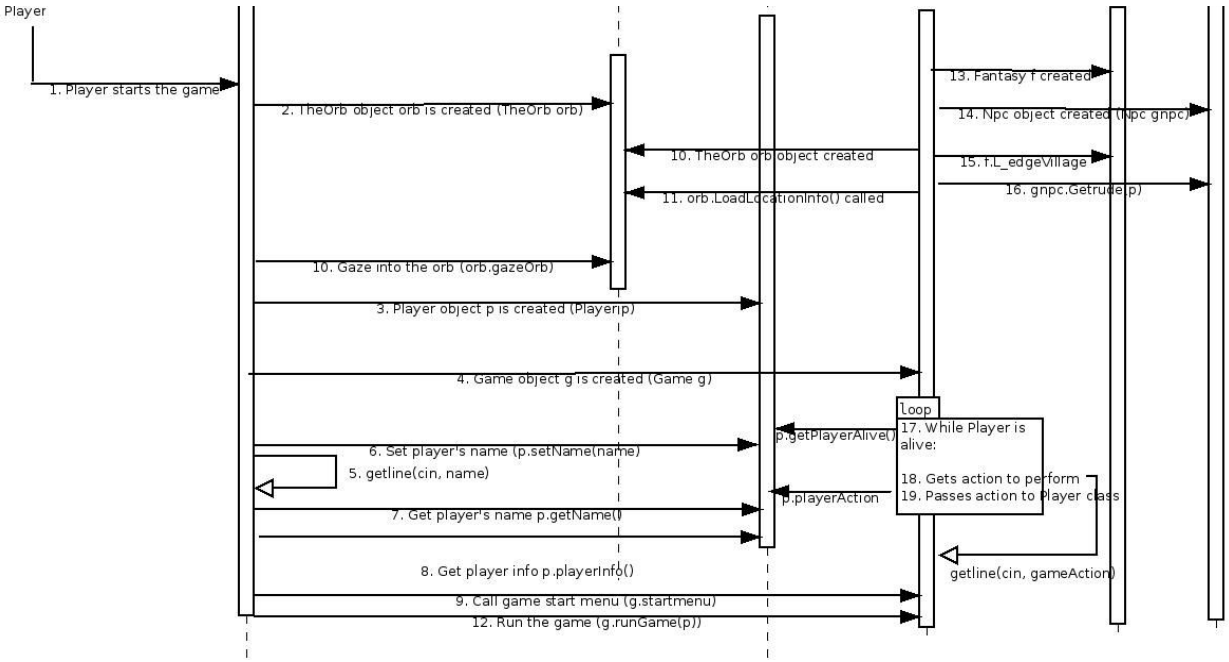

Figure 3 - sequence diagram showing the general progression through the game

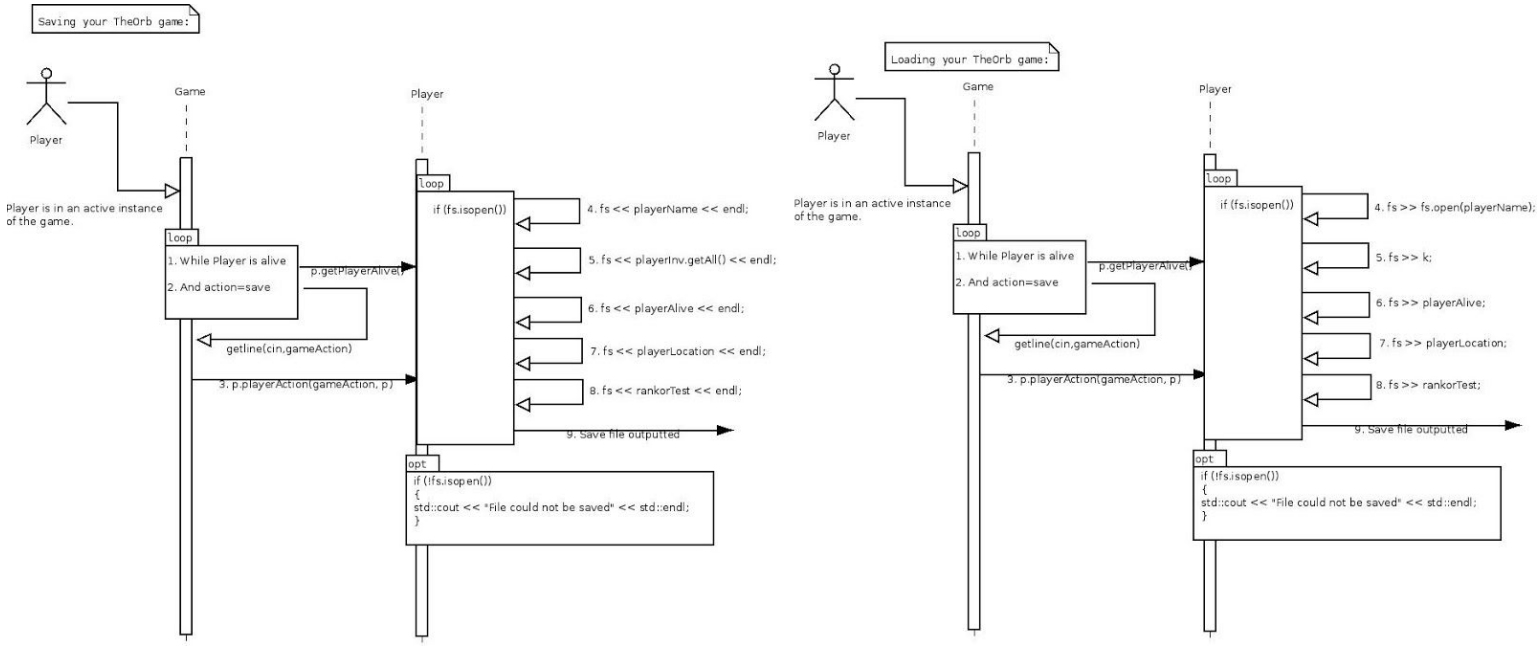

Figure 4 - sequence diagram showing loading and saving the game