Product Planning

The Cave of Caerbannog

By
Team MIGI2:
Taico Aerts
Chiel Bruin
Bram Crielaard
Wytze Elhorst
Robin van der Wal

Context project 2015-2016

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE.

Instructors:	prof. dr. A. Hanjalic and dr. A. Bacchelli
Context coordinator:	dr. ir. R. Bidarra.
Student assistants:	S. van den Oever and J. van Schagen

May 2016

Table of contents

1. Introduction

This document is the Product Planning of the game “The Cave Of Caerbannog”. The purpose of this document is to create requirements for our game and to put all these in a schedule.
To do this we will start off with a high level product backlog, which will list all the requirements of our game grouped by importance as well as a roadmap of when we want to have these requirements implemented.
Next is the product backlog which starts off with user stories, describing users in scenarios and what should happen in those scenarios, followed by a release plan which is an overview of the different releases of the game and which features it will have at these releases.
Finally we have the definition of done. This part, as its name suggests, describes when we deem something as done. This part is further divided in three parts describing the definition of done for each of these: backlog items, sprints and releases.

2. Product

2.1. High level product backlog

This section contains all the features we want to have in our game, using the MoSCoW method (H. van Vliet, 2008) to determine their importance, which orders the features as follows:
Must have - These features are mandatory for the game to function.
Should have -The game should have these features to be fully enjoyable, but will still function without them.
Could have - These features would be a nice addition to the game, but would mostly be a luxury and will only be implemented if there is enough time to do so.
Won’t have - These features will not be worked on during this project, but they might be implemented in a later project.

Must have:

- The Oculus Rift user must be able to see the cave in 3D on their Oculus Rift.
- The Oculus Rift user must be able to walk through the cave in a first person view.
- The other players must be able to see the cave in 2D and interact with it from their mobile devices.
- The helpers must be able to see the explored areas of the maze.
- The helpers must be able to spot nearby items such as keys and bombs in the cave.
- The helpers must be able to spot obstacles and help the Oculus Rift user avoid them.
- The Oculus Rift user must be able to pick up bombs and put them down again.
- The Oculus Rift user must be able to pick up keys and use them to open doors.
- The antagonists must be able to place bombs and plant landmines.
- The antagonists must be able to see a map of the entire cave.

- The antagonists must be able to spawn enemies that will chase the Oculus Rift player when they spot him.
- The helpers must be able to distract the enemies and lure them away from the player.
- The Oculus Rift user must be able to reach the end of the cave with the help of the helpers.
- The antagonists must not be able to make it impossible for the other players to win.
- The antagonists must be able to kill the rift user by hitting him with bombs, mines or enemies.

Should have:

- The Oculus Rift user should be able to activate “light switches” to give the helpers more vision.
- The players should have a limited amount of time to complete the cave.
- The Oculus Rift user should be able to move boxes or rocks.
- The Oculus Rift user should be able to climb boxes or rocks in order to reach a higher level.
- The helpers should be able to move certain platform and open gates to help the Oculus Rift user.
- The enemies should be able to die by walking on landmines or in range of bombs.
- There should be a projected overview of the cave that shows a big image of the cave for the helpers and the antagonists. This map does not give away information that has not yet been found by either the Oculus Rift user or the helpers.
- The game should have a soundtrack.
- The game should include ambient sounds.
- The game should have a way to change the difficulty by changing the time limit and the number of rooms the cave has.

Could have:

- The caves could be able to be randomly generated using seeds.
- The Oculus Rift user could be able to find a weapon to fight enemies with.
- There could be hidden rooms that the Oculus Rift user can find with bombs that offer bonuses to Oculus Rift user and his or her helpers.
- The game could have tutorial elements to help players get familiar with the game mechanics.
- The ability to load in your own textures and skins.

Won’t have:

- Multiple Oculus Rift users that compete against each other.
- The ability to create your own levels.

2.2. Roadmap

We will use the following planning to develop our game
Week 1: The concept stage.
During this week we set up the project and came up with concepts for our game.
Week 2 (Sprint 1): Working out the final concept.
During this week we pick out the best concept. Then we further flesh out this concept and use that as the definite concept of our game. Also we get familiar with the JMonkey engine.
Week 3 (Sprint 2): Start of development.
Now with a fully fleshed out concept, we can start on the actual development of the game. During this sprint we will start working on the basis of our game, such as generating a cave and being able to walk through it.
Week 4 (Sprint 3): Shaping the game.
During this week we will implement some of the more basic unique features of our game. Adding objects such as keys and bombs to our game, and implementing an interface for the helpers and antagonists.
Week 5 (Sprint 4): First playable spikes.
This week we plan to have the first playable parts of our game ready. This means that most of the basic features should be implemented which should allow the game to be playable to a certain extent and thus the first play testing can occur.
Week 6 (Sprint 4): Advanced shaping.
During this week we want to have all primary features (must haves) of our game implemented, so that we can start working on additional features. So this week will be both finishing our more advanced primary features (such as enemies) as well as implementing the basis of some secondary features.
Week 7 (Sprint 5): Release of the beta.
During this week we plan to release the beta. This means that we want to have a playable version of the game containing all of its basic functions. So this week will be spend on both creating that playable version as well as adding more secondary features to the game.
Week 8 (Sprint 6): Features, bugs and balance.
Now that we have a playable beta, we should be able to playtest the game extensively. This means that we will be able to find possible bugs, as well as imbalances in our game. During this week we will try to fix all the bugs we find and balance the game as much as possible. Aside from that we will still be working on implementing more features.
Week 9 (Sprint 7): More of the above and a trailer.
This week we will start working on the script for the trailer and we will start working on the production of the trailer as well. The final features will have been implemented by now, so during this week we will playtest the version of our game containing all features, so that we can remove bugs and balance the game for its final version.
Week 10 (Sprint 9): The final version.
During this week there should be no more implementations and the only changes to the code should be either bug fixes or balance changes, but even these should be finished at the start of this week. Once this is done, we will have the final version of our game. We will also finish the production of the trailer and showcase it alongside our game at the final presentation.

3. Product backlog

3.1. User stories of features

As the Rift player, I can look around the cave in virtual reality.

As the Rift player, I can move around in the maze.

As the Rift player,
When I pick up a bomb,
I am able to put it down again and the bomb will explode shortly after.

As the Rift player,
When I have picked up a key of a certain color,
I am able to open a door of the corresponding color.

As a non Rift player, I am able to see the cave in 2D.

As a helper, I am able to see obstacles that are invisible to the Rift player.

As an antagonist, I am able to place bombs near the Rift player that explode after a short time

As an antagonist, I am able to spawn enemies that will chase the Rift player.

As a helper,
When an antagonist has spawned enemies,
I am able to distract the enemies and lure them away from the Rift player.

As a bomb,
When I explode when the Rift player is near me,
The Rift player must be damaged and may possibly die.

3.2. User stories of defects

At the moment, there are no defects applicable.

3.3. User stories of technical improvements

At the moment, there are no technical improvements applicable.

3.4. User stories of know-how acquisition

As a player,
When I have played the game a few times,
I am familiar with all game mechanics.

As a player,
When I have never played the game before,
I will quickly pick up on how the game works.

3.5. Initial release plan

The initial release plan is heavily based on the sprints and the roadmap in Chapter 2.2. There is a release every week, which addresses another milestone.

Week	Milestone
1	Choose game concept and create requirements.
2 (Sprint 1)	Expand the game concept and familiarize with JMonkey
3 (Sprint 2)	Players can now perform basic interactions with the maze
4 (Sprint 3)	Players can now perform some more essential advanced actions (like spawning enemies)
5 (Sprint 4)	The game will be playable to a certain extent
6 (Sprint 5)	Players now have additional interactions with the maze (boxes, moving platforms)
7 (Sprint 6)	The game enters its beta, this means the game is playable
8 (Sprint 7)	The game will get balanced and any bugs will be fixed
9 (Sprint 8)	The trailer of the game will be developed, as well as the final version of the game
10 (Sprint 9)	The game is released (including the trailer)

4. Definition of Done

4.1. Backlog items

A backlog item is done, when the pull request(s) corresponding to the item comply with the following requirements.

- **Code**
  - Code has to be tested as thoroughly as possible. Line coverage, as measured by cobertura, has to be at least 80%. If this task is impossible, an explanation must be given as to why this is the case.
  - All methods and classes must have a javadoc comment explaining their functionality.
  - The code must result in a succeeding build on the continuous integration system, Travis. This implies that all junit tests in the code must pass.
  - The code may not contain warnings generated by checkstyle, FindBugs and PMD. If this task is impossible, an explanation must be given as to why this is the case.
- **Naming**

    - The name of the branch must fit with the contents of the branch. The following prefixes must be used to indicate the category:
        - “fix-” This branch fixes a certain issue or problem.
        - “feature-” This branch contains a new feature.
        - “doc-” This branch regards documentation.
        - “test-” This branch regards tests.
        - “balance-” This branch regards balancing of the different gameplay mechanics.
- **Pull Requests**
    - Every pull request must be reviewed by at least two people, excluding the user who created the pull request. The people that review the pull request will leave comments to indicate if the pull request is ready to be accepted, or if there are issues that need to be resolved first.

4.2. Sprints

A sprint is done when:

- All user story features from the sprint have been closed. If a user story is not finished an explanation has to be given.
- All the tests of these features have passed.
- All the documents on that sprint plan have been completed.
- There are no fatal bugs or errors in the code.
- The build passes.

4.3. Releases

A release is done when:

- All ‘must have’ features have been implemented and tested.
- Most (preferably all) ‘should have’ features have been implemented and tested.
- All documentation is in order.
- The trailer has been completed.

5. Glossary

Checkstyle

A static analysis tool which determines if the code complies with the style rules agreed upon by a development team.

FindBugs

A static analysis tool which detects possible bugs in java code.

PMD

A static analysis tool which uses a rule-set to determine if the code is erroneous.

Pull request

A request to add or change code and deliverables.

A. References

Hans van Vliet, *Software Engineering: Principles and Practice*, third edition, Wiley, Chichester (UK), 2008, p. 63