**CS673S16 Software Engineering** 

**Team 1 - TankInATank**

**Project Proposal and Planning**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Baoxiang Yang | Team Leader | *Baoxiang YANG* | 09/20/2017 |
| Haotian Wu | Requirement Leader | *Haotian Wu* | 09/20/2017 |
| Shuhan Liu | QA Leader | *Shuhan Liu* | 09/20/2017 |
| Lei Yang | Implementation Leader | *Lei Yang* | 09/20/2017 |
| Qiwei Zheng | Environment and Integration Leader | *QZ* | 09/20/2017 |
| Linlan Chen | Configuration Leader | *Linlan Chen* | 09/20/2017 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_3whwml4)

[Related Work](#_2bn6wsx)

[Detailed Description](#_qsh70q)

[Management Plan](#_2et92p0)

[Process Model](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.27177f40uci)

[Risk Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.a4oqwntk3mw)

[Monitoring and Controlling Mechanism](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.ywdoc2clc9yt)

[Schedule and deadline](#_1t3h5sf)

[Quality Assurance Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.72e1f4uawy2r)

[Metrics](#_3as4poj)

[Standard](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.vc72k6dweldv)

[Inspection/Review Process](#_35nkun2)

[Testing](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.r5d5mhtlf0kq)

[Defect Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.54a4wuncjg1c)

[Process improvement process](#_1pxezwc)

[Configuration Management Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.hw41vg4ykxen)

[Configuration items and tools](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.bwlb4d4vdox2)

[code commit guidelines](#_4i7ojhp)

[References](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.8mva2050iy7t)

[Glossary](#_1ci93xb)

# Overview

In this project, we will build a 2.5D tank fighting game on a desktop with Unity3D. The video game industry has been continuously growing since the beginning of the twenty-first century. In recent years, with the development of GPU design, computer graphics algorithms, and virtual reality, the potential has been unleashed. Lots of indie games have been made by small companies and groups. We think it will be a good practice if we can make a well-designed indie game. The game will be an offline game that allows two players to control two tanks and fight against each other in a variety of environments. Namely, this game will be a local multiplayer game that let two players use a single keyboard. We don’t see any recent 2.5D tank game has this vintage play mode that is fun to play with friends sitting side-by-side. It’s a fantastic play mode that you can have fun with family and friends in a party, so we decided to make our own. 2.5D game is a kind of game that visualizes the scene in 3D but has the characters moving on a 2D plane. The camera will have a fixed angle looking towards the plane and the characters only have 4 direction of movements. Moreover, we will try to make an online multiplayer version of it and add AIs if we have time in a later iteration. This project will be written in C# using the platform provided by Unity. Also, a backend will be build once we exceed offline only version.

# Related Work

There are countless tank fighting games that you can find on steam or other gaming website, however, as stated above. There is no recent keyboard-controlled 2.5D tank fighting game that has offline multiplayer mode. Most of the related 2.5D tank game were made more than a decade ago.

Related Games:

* Tank Game
* Tank Destroyer
* Hogs of War
* World of Tanks
* ToonTank
* S.W.I.N.E.
* Recent Local multiplayer game
  + Gang Beast
  + Rocket League
  + Portal

# Proposed High level Requirements

Pivotal tracker link: https://www.pivotaltracker.com/n/projects/2112728

* 1. Functional Requirements
     1. Essential Features
        1. Start game with another player on the same screen.
        2. Each player can pause, restart or quit the game during the game.
        3. The two players can choose one map out of three.
        4. Both of the players can make the tank move left, right, up or down.
        5. Let the tank shoot shell and if a shell hits a tank, tank will lose HP..
        6. Tank has HP and game will be over if one tank loses all HP.
        7. Tank will have areas(mountains, etc) where cannot move through.
     2. Desirable Features
        1. Choose different types of tank.
        2. Use items to recover HP, increase attack or speed up.
        3. Can destroy some objects(walls, hills, etc) on the map by shooting shell.
        4. The explosion of map destruction deals damage to tank.
        5. Have more than 5 maps to choose.
        6. Let system randomly choose one map.
     3. Optional Features
        1. Can let system generate a map randomly.
        2. Can play the game alone.
        3. Can play the game with other players online.
        4. When playing the game alone, computer will control another tank. AI will be designed.
        5. Can play with a specified player when playing the game online.
  2. Nonfunctional Requirements
     1. There should be a easy understand menu for users with no more than 5 buttons.
     2. The tank should be able to turn around in 6 seconds to make give the user a smooth game experience.
     3. To make sure map size is reasonable, the tank should be able to move across the tank within 20 seconds.
     4. Detailed game instructions.
  3. Implemented Features (to be completed at the end of project)
     1. Start game with another player on the same screen.
     2. Each player can pause, restart or quit the game during the game.
     3. The two players can choose one map out of three.
     4. Both of the players can make the tank move left, right, up or down.
     5. Let the tank shoot shell and if a shell hits a tank, tank will lose HP..
     6. Tank has HP and game will be over if one tank loses all HP.
     7. Tank will have areas(mountains, etc) where cannot move through.

# Management Plan

# (For more detail, please refer to SPMP document for encounter example)

## Process Model

* + 1. Basic work process model: Scrum
    2. Tools would be used for Scrum:
       1. Trello or pivotalTracker
    3. Roles management:
       1. Product Manager: 1
       2. Developers: 6
       3. Testers: 3
       4. Documents: 1
       5. Q&A: 1
       6. Settings: 2
    4. Team Communication:
       1. By weekly meeting
       2. Monday’s work from home
    5. Team Assess:
       1. By writing peer reviews on midpoint and final.

## Objectives and Priorities

* + 1. Basic goal: Finish build the Unity 2.5D tank game with a basic map and manipulate by keyboard input.
    2. Stretch goal:
       1. Map-Select function(Multiple maps or randomly generated maps)
       2. Import skills with different tank model (Increasing fun)
       3. Online mode (Support for internet battle)
       4. Multiple players mode (Based on online function)
       5. Import AI (Support for agent AI)
       6. Deep learning (AI Tanks can learn by battles)
    3. Potential Tasks:
       1. Iteration 0:
          1. Documents(high)
       2. Iteration 1:
          1. Configuration Setting(high)
          2. Environmental Setting(high)
          3. Build for basic map function(medium)
          4. Build tank model and basic movement logic(medium)
          5. Documentations(medium)
       3. Iteration 2:
          1. Online mode(low)
          2. Multiple maps(low)
          3. Demo test(low)
       4. Iteration 3:
          1. Final demo test(low)

## Risk Management (need update constantly)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Risk Type | Risk Description | Risk Sign | Risk Impact | Risk Response | Person Responsible |
| 17001 | Range | Project Definition too wide. | Cannot determine a durable vision | May occurs useless code | Set a temporary meeting to get a quick durable vision | YBX  WHT |
| 17002 | Process | Cannot finish tasks on time | Key functions lay off time like Block on Hit logics | Block project process and other people Waste time. | May reassign the task to another person and record.(Would through weekly meeting or communication tookls) | All p |
| 17003 | Quality | Too many bugs | Too much Test failures | Waste time. | Code reviews other peer to make sure everything is good. | All  CLL |
| 17004 | Tech | Bugs are hard to solve | Stuck for hours on a single steps | Potentially need to replace the whole function | Need more than 2 peers to have a quick discussion | All |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## Monitoring and Controlling Mechanism

* + 1. Documentation Management Tools:
       1. Google Docs; Google Drive
    2. Coding management Tools
       1. Github
       2. Pivotal Tracker
    3. Communication Tools:
       1. Slack
       2. Wechat
    4. Others:
       1. Weekly meeting

## Schedule and deadlines (need update constantly)

* + 1. Iteration 0 Presentation: 09/27
       1. Finish SPPP: 09/25
    2. Iteration 1 Presentation and demo: 10/18
       1. Configuration Setting: 09/29
       2. Environmental Setting: 10/02
       3. Basic Map Design: 10/10
       4. Tank Model and basic movement logic: 10/12
       5. Slides and demo test: 10/16
    3. Iteration 2 Presentation and demo: 11/08
    4. Iteration 3 Presentation and demo: 12/06

# Quality Assurance Plan

## Metrics

* + 1. Definition  
       Metrics 1: Number of defects existed in module (Stability)  
       In the end of each iteration, the number of defects found by QAs will be counted.  
         
       Metrics 2: Code lines to test cases (Stability)  
       In the end of each iteration, code lines to test cases will be counted. This number should be as small as possible. Which proves we have enough test cases.  
         
       Metrics 3: Test coverage (Stability)  
       In the end of each iteration, test coverage calculator will be used to analyze the test coverage, and documented.  
         
       Metrics 4: Number of classes (Complexity)  
       In the end of each iteration, number of classes will be counted and documented.  
         
       Metrics 5: Code lines (Complexity)  
       In the end of each iteration, number of code lines will be counted and documented.  
         
       Metrics 6: Number of files (Complexity)  
       In the end of each iteration, number of files will be counted and documented.  
         
       Metrics 7: Number of working hours per week per person (Workload)  
       In the end of each iteration, number of working hours will be counted using burndown chart.  
         
       Metrics 8: Number of code lines per person (Workload)  
       In the end of each iteration, number of code lines per person will be counted using burndown chart.  
         
       Metrics 9: Defect fix rate (Quality)  
       In the end of each iteration, number of defect fix rate will be counted to ensure quality.
    2. Results (to be completed at the end of each iteration)

## Standard Design Documentation standard The programming documentation should include class description. Each API description must have parameter, return, description. Each design document must be signed by every team member Each iteration design document can’t exceed 30% change compare with the previous iteration Quality Assurance Documentation standard The quality assurance documentation will based on a iteration mode. We will use pivotal tracker to track some significant bugs, and discuss about the progress about small bugs in weekly meeting and recorded in meeting minutes. Also, all the test cases related. Also include the related metrics for stability and quality. Again, since we are using iteration mode, the documentation might change very fast. Coding standard Function line standard Each function has maximum 40 lines. Row letter maximum standard Each line can’t exceed 100 letters. Recursive standard Recursion function can’t call another recursion function Unit test standard Unit test coverage must be over 80% for each methods. Type standard Each type must be final for class member. Static type is not allowed for function class. Name standard Method and variable naming must follow Camel naming standard

## Inspection/Review Process

**Code Review** When: After the submission of each code review  
 Who: Programmers  
 The code review process focuses on the code quality and primary functionality. Each code change will be reviewed by the rest team on Github. Reviewers and reviewees may have dialogs where the code was changed.  
   
 **Requirements Review** When: End of iteration demo  
 Who: All  
 During the development process, the requirements might change based on customer feedback (the feedback from demo). We will review the feedback and make updates.

**Documentation Review** When: End of iteration demo  
 Who: All  
 All team members will be involved in the documentation review at the end of each iteration.

## Testing

## (e.g. who, when and what type of testing to be performed? How to keep track of testing results?)

A separate document about testing result should be linked here.

1. Unit tests and system tests:

As a video game, apart from the regular functions, the most important thing is to get rid of bugs. Since most bugs are caused by boundary situations, we need to implement **Boundary Value Testing (BVT).** Boundary Value Testing is a testing technique that is based on concept “error aggregates at boundaries”. In this testing technique, testing is done extensively to check for defects at boundary conditions. We will do this at both unit level and system level. However, we found the unity test tool is deprecated, and no longer available in store, we will do the testing without tools.

1. Acceptance tests:

Another method we will implement is **Beta Testing**. Like many video games have beta versions, the best way to test a video game is to ask players to play it. Our team members will all play the beta version of this interesting game, to find if there are other test cases not covered by BVT. We will use these method to cover acceptance tests.

1. The three kinds of tests will be implemented at the end of every iteration after the game have basic functions and can be played.
2. To keep track of testing result, we will record the result of every corner case, and summarize them to find out what is the cause of a specific problem.

## Defect Management

(e.g. describe the criteria of defect, also in terms of severity, extend, priority, etc. The tool used to management defect, actions or personnel for defect management).

1. Criteria of defect: Priority of a defect would indicate the urgency with which it would need to be fixed. As a game, the most severe problems are bugs that will affect the justice of the game, such as errors in calculating the health bar of the tank or the wrong damage. Other things like whether the tank is beautiful or not are considered as less severe and low priority.
2. To manage defects, we use pivotal tracker to record significant bugs. Therefore every teammate can see what the error is and programmers in our team can work on it and response changes to testers and PM.

# Configuration Management Plan

(For more detail, please refer to SCMP document for encounter example)

## Configuration items and tools

* + 1. IDE: Unity 2017 1.1
    2. Version Control: Git
    3. GitHub: <https://github.com/LinlanGrace/MET-673-Team-1>
    4. Environment: <https://docs.google.com/document/d/1P9i-BCCFUyaJALhz_qvIrqTUFOByBwylM56zpwCNJdE/edit>
    5. Game Engine: Unity 3D
    6. Pivotal Tracker: https://www.pivotaltracker.com/n/projects/2112728

## Change management and branch management

We have three kind of branches:

* Development Branch: All the development are in this branch.
* Feature Branch: Developers should feel free to add feature branches for specific features.
* Master Branch: Product version.

Other details about merge operations can be found in Git Tutorial.

## Code commit guidelines

* Every time a change is to be made, you should pull from mainline first and resolve all conflicts before pushing any code.
* Commit steps:
  + $git add <new file>
  + $git commit -m “comment”
  + $git push
* Code should be committed with clear style and documentation.
* Only after reviewed by one or two of team members can the code in development branch be merged into master branch.

# References

# Glossary

* Indie games: An independent video game (commonly referred to as an indie game) is a videogame that is often created without the financial support of a publisher, although some games with publisher funding are still considered indie. Indie games often focus on innovation and rely on digital distribution. (Wikipedia)

# Local Multiplayer Games: Games can be played by multiple users on a singular device