Project Management Plan

Grand Slam Baseball All-Stars 2: Electric Boogaloo

COP4331C, Spring, 2018

Team Name: Team MLB Card Game 2

Team Members:

- Daniel Carman
- John DeSimone
- Malik Henriquez
- Ronald Marrero
- Brian Wengier
- Kaleb Yangson

Contents of this Document

Technical Progress Metrics

Plan for tracking, control, and reporting of progress

Project Overview

The objective of our project is to create a Unity based game that utilizes a database of baseball cards to let players manage their own dream team of players and battle against others through asynchronous online play. Players will create their team by opening card packs that contain players with unique stats, items (such as different bats, balls, and gloves, each with their own stat modifiers), and temporary stat boosters, and then construct a 10-man roster (five batters, 3 base-men, 2 outfielders) from their collection of cards. Once a team is created, the player will be able to battle teams previously created by other players. The gameplay will focus on team management and item collection, with battles being incredibly quick innings of baseball based on team stats and random number generation. Since the gameplay is built around team management, stat management, and random number generation, there will be no need for both players to be online to play; one player will simply initiate a battle and then be matched with another team stored on the database, similar to the popular NBA 2K17 mobile game.

Applicable Standards

- Our team will be strictly following the Unity C# coding guidelines, as it will allow for ease of code management for anyone familiar with unity, were the game to be sold to another entity at some point. These guidelines are as follows:
    - Braces are mandatory for all control flow statements and must begin on the line following the control flow statement and end the line immediately after the block has concluded. The only exception to this is single line statements, where open and closed braces can appear on the same line.
    - Indentation for each level of a series of control flow statements is mandatory (including for cases in switch statements). Indentation must be a single tab character.
    - Commenting is mandatory for each class and function and should describe what a class is used for or what a function's inputs, purpose, and outputs are. Commenting is encouraged throughout code to explain logical flow. Comments are preferred to be above code rather than to the right of it.
    - In function declarations and calls, a single space must be placed after each comma to separate parameters and nowhere else. This means omitting spaces after the function name but before the parentheses and spaces within the parentheses before or after all parameters are declared.
    - A single space must be used before parentheses in all control flow statements, as well as a single space before and after comparison operators.
    - For all member variables, parameters, and local variables, camelCasing is required. Functions, properties, events, and class names must use PascalCasing. Interface names must be preceded with the capital letter "I".
    - Source files should contain only one public type, and the file and type should share a name. Class members should be organized alphabetically and placed in groups based on what they are (a group for methods, a group for constructors, etc.)

  More information on the Unity C# coding guidelines can be found here:
  http://wiki.unity3d.com/index.php/Csharp_Coding_Guidelines

- Our team's artifact size metric standards are threefold, focusing on number of classes, number of assets, and number of deliverable reports. The breakdown of these artifacts is as follows:
    - Number of Classes - The expected number of classes for our project is 13, with 5 classes being produced for the front end, and 8 classes being produced for the backend. For the front end, we expect to make a class for the main menu, the primary in-game interface, the pause menu, the pack store (for purchasing new players and items with in-game currency), and one for the option menu. For the backend, one class will be made to give functionality to each of the front end classes. Additionally, there will be a back end class for the in-game logic, a class to handle the various kinds of random number generation in game (for handling probability of success on a hit, probability of getting rare cards, etc.), and a class to interface the game with the database containing player information.
    - Number of assets - Assets for the game are broken up into 3 categories; card assets, sound assets, and animation assets. The total number of expected card assets, including cards for both players and items is expected to be around 100 at a minimum, going up to around 200 if time allows. At least 1 sound asset and at least 1 animation asset must be present in the final project, but this number may go up to around 5 if time allows.
    - Number of deliverable reports - The number of deliverable reports for the project is expected to reach 68, including those listed in the deliverables section, as well as 10 progress reports from each of the 6 team members that will be delivered to both the project manager (Dan) and the CTO (Safa) on a weekly basis.

Project Team Organization

In order to increase productivity, we have divided the group members listed above into two groups. The project manager Dan Carmen, along with John DeSimone and Ronald Marrero will work on the backend logic for the deliverable game, while Malik Henriquez, Brian Wengier, and Kaleb Yangson will focus on front end development.

Individual tasks will be assigned through Trello by Dan Carmen. Daily scrum participation on the group Slack channel is required by group members. In addition, a weekly big picture meeting which will take place in an online voice chat or in a face-to-face meeting on Friday in order to give status updates on big picture tasks, keep track of timetables for the project, and to more effectively allocate human resources as the project continues.

Deliverables

Artifact	**Due Dates**
Individual Weekly Progress Reports	Weekly (Fridays) submission throughout the semester through WebCourses
Concept of Operations	2/9/2018
Software Project Management Plan (SPMP)	2/18/2018

Software Requirements Specification (SRS)	2/25/2018
High-Level Design	3/18/2018
Detailed Design	3/21/2018
Test Plan	4/5/2018
Test Results	4/9/2018
Source, Executable, Build Instructions	4/9/2018

Software Life Cycle Process

For this project, the development team has decided to use the Scrum software development methodology, which is a combination of incremental, agile software development framework . This development methodology divides work amongst the respective programming teams into timed milestones called 'sprints.' Progress is tracked and milestones are updated in short, daily meetings called scrums. The development team chose this methodology because it was designed specifically for a programming team of our size (typically three to nine members) and allows the team flexibility in allocating resources and accomplishing the overall programming deliverable by in the assigned development time.

Tools and Computing Environment

The programming team is developing this deliverable for release on Android supported devices. Development will take place on a mix of PCs and Mac laptops. In addition, the Unity engine will be used in development along with C# as a scripting language along with possibly UnityScript (Unity's Javascript variant) using Visual Studios. Product testing will be performed on Android emulators and Android phone systems.

Configuration Management

Version Control will be accomplished through a private Github repository. Only members of this project group will have access to view the project or create pull requests. There will only be two branches to track our development: QA and Master.

The QA branch will not have strict acceptance requirements. If it builds, it will be merged. This branch will be the one that all pull requests go through. This will house any revisions with untested features that will not affect the master branch.

The Master branch will store only stable builds. These builds will only come from the latest stable build in the QA Branch. Once a QA build has been thoroughly tested, it will enter a change control process to determine if it will be merged to master. The entire team will participate in the change control process

to decide if the build should be merged. If the decision is made, the QA version will simply be merged up to Master as well.

Ronald Marrero will serve as the GitHub administrator and will be responsible for merging pull requests as well as restoring previous versions in the event of some failure. Ronald will act in accordance with the group's decisions instead of making autonomous decisions.

Quality Assurance

Quality Assurance will take place at multiple points during development. As code is completed it will be pushed to GitHub. Then, a developer who did not write the code will be chosen from a rotation and review the code that has been submitted for peer review purposes. If problems are found, the code will be bounded back to the original developer along with notes detailing necessary edits to the code.

As the project gets further into development, an Excel sheet, managed by John DeSimone, will be created to keep track of tests that need to be executed. Successful tests will have their status cell colored green, while failures will be colored red, as well as, have subsequent cells filled out with details of bugs that have occured. Bugs that have been addressed will have their status cells color changed to yellow, in order to signal for a retest. This testing suite should be run for each new release of the application.

Risk Management

The following are our team's currently expected risks and our plans to mitigate them. The probability of each risk is given as a percentage from 0 to 1 (with 0 being impossible and 1 being guaranteed), and a risk factor ranging from 1 to 5 (1 being minimally impacting, 5 being catastrophic)

- A team member leaves the group - Weather it's due to a team member's non participation or unforeseen circumstances, it is possible that a team member could be removed from the project group. To lessen the impact of this risk, our team shall work diligently to complete deliverables ahead of time and strictly adhere to the project's coding standards so that in the event a group member does leave, there will be less work to do and more information to go off of for the team members that have to pick up where they left off.
    - Probability: 0.1, Risk Factor: 3
- A new team member joins the group - If another team loses too many members, there is a possibility that another person could be reassigned to our project group if their previous project is deemed unviable. To lessen the impact of this risk, thorough commenting shall be peppered throughout all code artifacts, and all paper artifacts shall adhere to the same document formatting.
    - Probability: 0.05, Risk Factor: 4
- Library Deprecation - In the event that a library becomes deprecated, code refactoring will be added to the list of assignments needed to be completed. The workload caused by this refactoring shall then be distributed evenly across the group. Aside from that, library depreciation is an accepted risk for the project, as our team has no way of predicting which libraries will go out of use.
    - Probability: 0.2, Risk Factor: 2
- Compatibility issues with team code - Since this project is being developed by multiple people, it is extremely likely that there will be compatibility issues when it comes time to packaging each

team member's artifacts together. In order to lessen the impact of this risk, all team members shall strictly adhere to the coding standards described above. Additionally, each artifact shall undergo peer review whenever a new version is uploaded to the git repository.
    - Probability: 1, Risk Factor: 1
- Github repository crash - Since we have no control over the stability of the Github repository being used to house all code deliverables for the project, there is a small chance that this repository could crash or become faulty. In order to lessen the impact of this catastrophic risk in the event it occurs, each group member will clone a copy of the master branch to their local machine whenever it is updated.
    - Probability: 0.05, Risk Factor: 5

Table of Work Packages, Time Estimates, and Assignments

Task Name	Jan '18	Feb '18	Mar '18	Apr '18	Person In Charge
Main Menu					Kaleb
In-game Interface					Brian
Pause Menu					Malik
Pack Store					Kaleb
Option Menu					Brian
Class Functionality					Dan
In-game Logic					John
Random Number Generator					Ron
Database Interface					Dan
Card Assets					John
Sound Assets					Ron
Animation Assets					Malik
Weekly Reports					Everyone

The GANNT chart shows the 13 different artifacts that the group will be working on. Task Name is the name of the tasks assigned, the dates represent the month and year, and Person In Charge represents what group member is responsible for making sure that the task gets completed. The blue bars within the graph represent the estimated amount of time that each task will take.

Technical Progress Metrics

Requirements Phase:

- Number of meetings done with group - This metric will benefit the team because throughout the project it will be necessary for the team to meet, face to face or conference call, to ensure everyone is on the same page.
- Number of requirements created/completed - This metric is important because the team will need to create requirements throughout the project.
- Number of requirements changed - This metric is important because as we progress in the project it will be necessary to look back and modify some of the team's previous requirements.
- Individual Progress Reports - This metric is important because it allows the project manager to keep track of the work of each project member.

Design Phase:

- Number of Classes designed - This metric will be important because during the design phase the team will be constantly creating new classes.
- Number of uml diagrams - This metric will be important because if design is going smoothly our team will be consistently making new diagrams for the system.
- Number of revisions completed - This metric is important because throughout the design phases there will be a need to make a revision in order to improve the system.

Implementation Phase:

- Lines of Code Per Day - This metric will be beneficial because it will ensure that project team members are actively working everyday on implementing the project.
- Number of classes completed - This metric will be important because it will ensure that the project manager has a general idea of the team's progress and makes sure no classes are missed during the implementation phase.

Testing Phase:

- Number of test cases - This metric is important because if testing phase is conducted correctly the project team will be testing multiple variables multiple times to make sure that the final product is reliable and of good quality.
- Passes/fails for tests - This metric is important because it will allow the team to keep track of the results of each test case throughout the testing phase of the project.
- Bugs found/fixed - This metric is important because during if testing is done correctly bugs will be found occasionally and the team will need to fix the problem.

Packaging Phase:

- Number of builds - This metric is important because during the packaging phase the team will need to conduct multiple builds to ensure a reliable final product.

Maintenance Phase:

- Number of updates - This metric is important because as Operating Systems are updated it is important to update the program to ensure it remains a quality product.

Plan for tracking, control, and reporting of progress

Each team member will post their individual progress report weekly through WebCourses: the weekly report should include time spent in each activity, completed task(s), tasks in-progress, tasks planned for the following week, and individual issues and problems.

Each week, each team member collectively read and analyze the logs; examine the technical content of the work done to date; examine the technical progress metrics; consider the QA results; reassess the potential project risks; and take corrective action if necessary.