**PROJECT TRIALS**

Design Document

TEAM 2

Garrett Kizior

Keith Droll

Nicolas Bratton

Ryan DeSalvio

**Table of Contents**

1. Purpose 4

1.1 Functional Requirements 5

1.2 Non-Functional Requirements 7

2. Design Outline 7

2.1 Architecture Outline 8

3. Design Issues 9

3.1 Functional Issues 9

3.2 Non-functional Issues 11

4. Design Details 12

2.1 Story Outline 12

2.2 Basic Character Class Outline 14

2.3 Item Class Outline 15

2.4 General Game Flow Outline 16

****

1. **Purpose**

Project Trials is a top-down action video game that incorporates elements of role playing and storytelling. The purpose of this project is to create a system that responds to player input and allows the player to complete objectives such as fighting against AI enemies, all the while letting the player choose how to complete these objectives.

To achieve this, we will implement a model-view-controller architecture for the software, including classes to represent the various aspects of the game such as the players and enemies, the items, and the world/level generation.

**1.1 Functional Requirements:**

As a user,

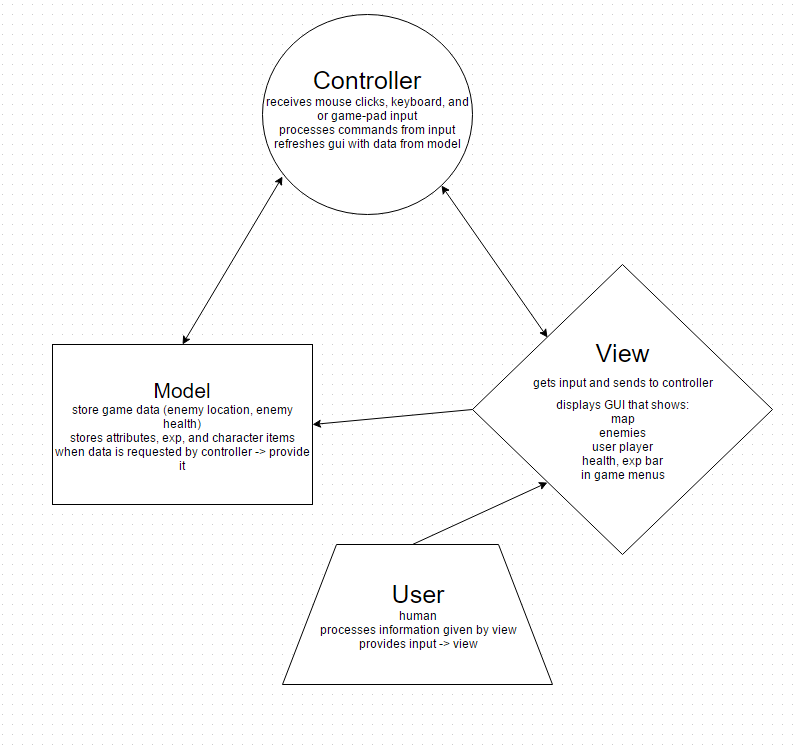
* I would like to start a new game
* I would like to pick up items
* I would like to pick up weapons
* I would like to be able to choose different dialogue options
* I would like to be able to view high scores
* I would like to know what my current health, experience bar, and level number is
* I would like to spend experience points on skills
* I would like to choose a perk after each level
* I would like to be able to use a controller or keyboard
* I would like my character attributes to be shown in a simple menu
* I would like to choose my difficulty level (if time allows)
* I would like to pause the game (if time allows)
* I would like to fight a final boss at the end of the game
* I would like to fight in either ranged and melee styles
* I would like to view an end-game statistics screen

As a developer,

* I would like to like to include animations for character movement
* I would like to like to include animations for character attack
* I would like to be able to store high scores
* I would like for enemies to spawn at an increasing rate as the level progresses
* I would like to randomly generate levels, maps
* I would like to make sure gameplay is balanced (enemy vs. player)
* I would like each level to have a different setting
* I would like the story to be entertaining yet unobtrusive to gameplay
* I would like the story to be a choose-your-own-adventure style story
* I would like the game to support multiple platforms (if time allows)
* I would like to implement an achievement system (if time allows)
* I would like to incorporate a start menu that allows for starting games, and viewing high scores, etc.
* I would like to allow for cooperative play (if time allows)
* I would like the game to run off an executable file
* I would like to implement different terrain objects that affect the character
* I would like for some enemies to follow the character blindly
* I would like for some enemies to shoot the character
* I would like to have a health system for enemy (not all enemies have the same amount of health)
* I would like to allow the character to equip armor as well as weapons (if time allows)
* I would like to include a credit scene at the end of game (if time allows)
* I would like to include a store to purchase character skins, etc. with in-game tokens earned from completing each play through (if time allows)
  1. **Non-functional:**
* As a user, I would like to have fast response times with user input
* We must be able to play this game on PC
* The menus and interface needs to be simple, intuitive and responsive
* We must have fast frame rates, and minimize computer resources
* Art style must be attractive and simple
* Game must include catchy and fitting soundtrack
* Game must include sound effects for different events
* Game is stored in an executable file – this prevents people from viewing source code

**Design Outline**

**2.1 Architecture Outline**



For our architecture design we chose the Model-View-Controller pattern, because we felt it represented most what we wanted to accomplish. This pattern integrates closest to our vision of the final product; we have the view which is the game interface, the model which is the backend where the game data is stored locally, and the controller which is the game code which retrieves and manipulates the game data based on player interaction within the game. Most of the work will be done in the controller part of the pattern, as that is where the code we will be developing come into effect.

1. **Design Issues**
   1. **Functional Issues**

**Functional Issue #1**: Since the game is going to be a game based

on repeated quick play-throughs, should the user be able to save his or his progress mid-game?

**Option A**: Yes, integrate a save game feature. Where player

can resume a game exactly where they previously were

**Option B**: No, Player must finish the play though without saved

progress in one sitting

**Decision**: Option B. Because saving a game would encourage

slowing the pace of the game down, when, we as the developers, intended the game to be based around quick individual playthroughs. Also allowing the feature of a saved game might encourage players to reload past saves to repeat a favorable playthrough.

**Functional Issue #2**: In game, when a player is to pick up an item,

how should this be done?

**Option A**: Player automatically equips whatever item he walks

over

**Option B**: Player only equips the item that is walked over if they

also press the required action button

**Option C**: Player automatically equips consumables when walking

over them, but weapons also require pressing an action button

**Decision**: Option C. Automatically equipping a weapon will

distract the player from the primary purpose of the game – defeating enemies. We don’t want the user to be afraid of picking up a weapon they don’t want. As far as consumable items, these items are passive and won’t swap out onto the playing field when collecting another consumable, so they should only be automatically equipped

**Functional Issue #3**: Should we provide a feature to view previous

local high-scores?

**Option A**: Yes

**Option B**: No

**Option C**: Only view high-scores at the end of a completed

playthrough

**Decision**: Option A. This will give players the ability to compare

previous runs and the ability to share scores with friends. We believe this will promote re-playability, self-to-self competition, and self-to-friend competition

**3.2 Non-Functional Issues**

**Non-Functional Issue #1:** What language and game development

engine are we going to use?

**Option A:** Unity engine with C#

**Option B:** Unreal engine 4 with C++

**Decision:** Option A. Because we have zero experience with

Unreal, and because our team has collectively some experience with Unity, we decided to go with Unity and C#.

**Non-Functional Issue #2**: What will we utilize for a soundtrack?

**Option A**: Create our own soundtrack

**Option B**: Purchase a soundtrack from 3rd party

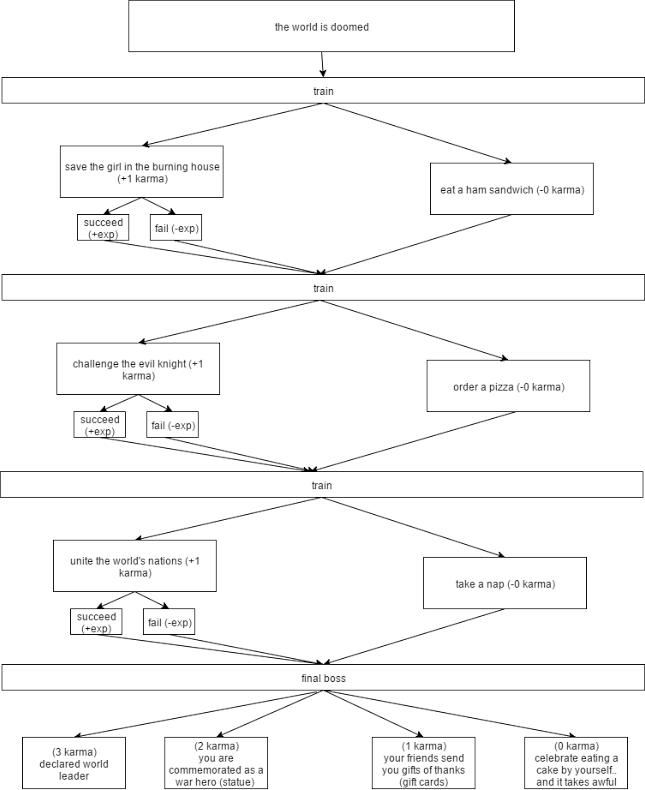
**Option C**: No soundtrack. Sound is solely from sound effects in-

game

**Decision**: Option A. We believe this will increase the originality of

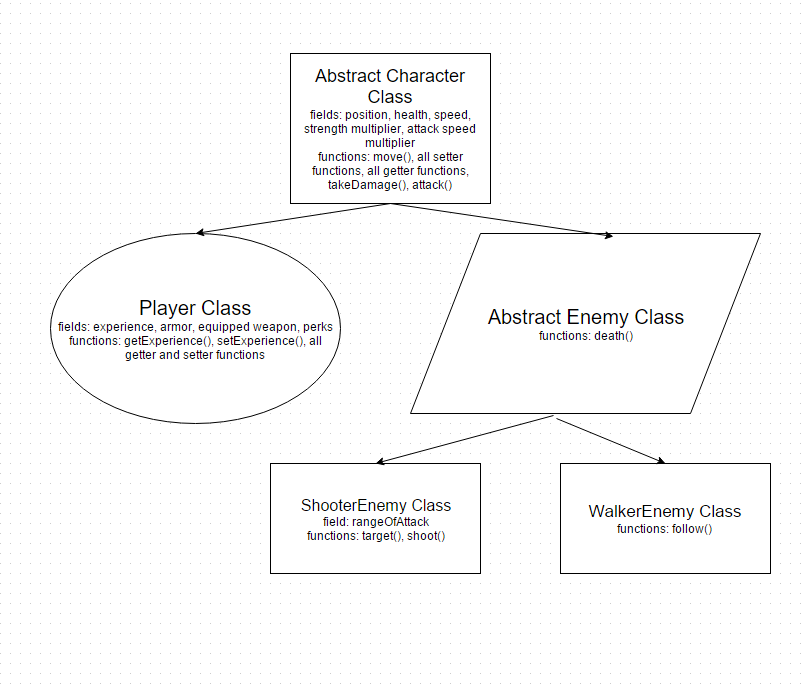
our game. We also believe that creating a soundtrack will allow us to create the music for the game, rather than create the game for the music.

**Design Details**

**4.1 Story Outline**

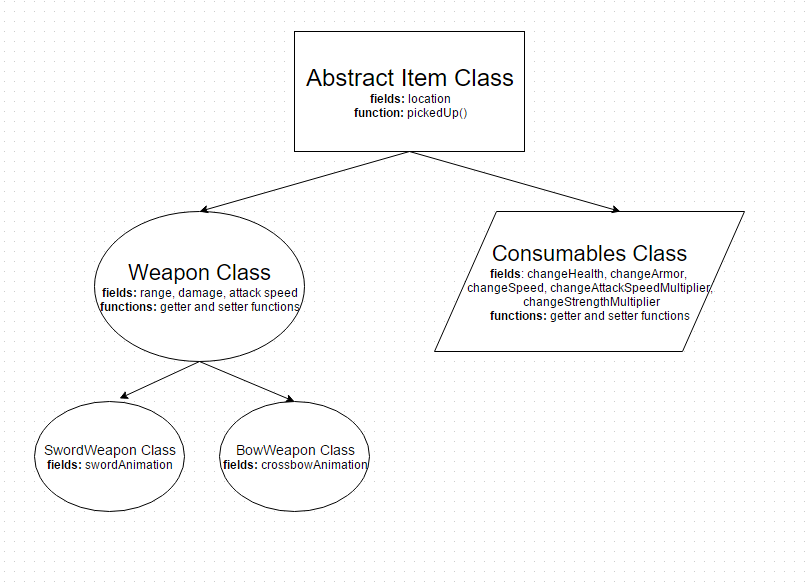
**Story Outline (cont.):**Our story diagram follows a simple outline. The beginning of the plot will be set up with a basic cut-scene using graphics and text giving the player some context and motivation for playing the game. The game will then progress into the first of several levels labeled as “train” in the graph. These levels are where the user will actually play the game and are meant to allow the user to progress through the level system and make decisions in the perk tree. Following the completion of a level, another cut-scene is displayed using, again, graphics and text to convey some advancement of story. The user is then presented with a choice between one of two options in the story that affect a karma meter allowing the user to possibly gain a bonus and affect the outcome of the story. This train of events happens 2 more times until a final boss is encountered and the player reaches an endgame which is determined by their karma level.

**4.2 Basic Character Class Outline**



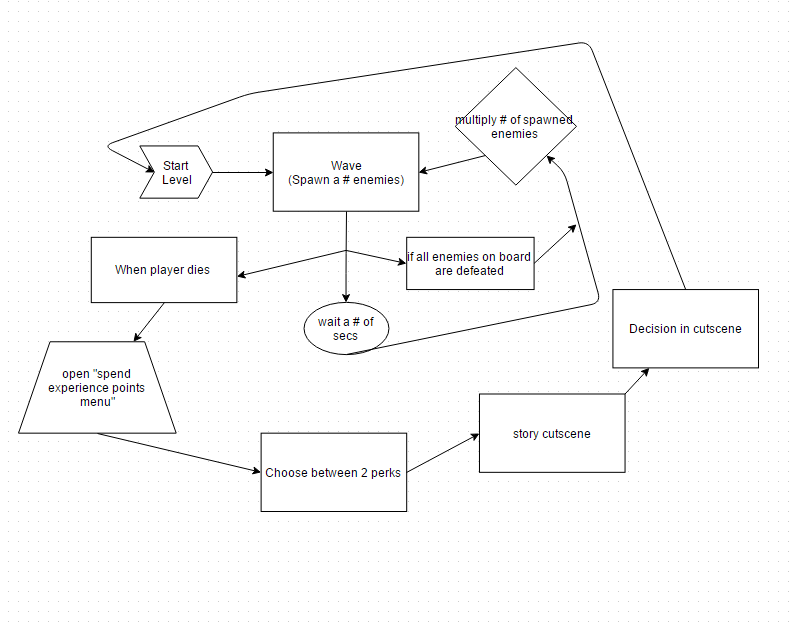
All actors/characters in the game belong to the Abstract Character class. A character is decided to be either a Player (the playable character that the User plays as) or an Enemy (the opponents faced by the User while playing). If the Character is a Player, they gain all of the attributes appropriate to the Player class, such as experience, equipped weapon, perks, among others. These are attributes that are unique to the Player class, and Enemy Characters may not have these same characteristics. Likewise, Enemies have variables that Players do not, such as rangeOfAttack and methods such as follow() and target(). There are other subclasses of the abstract Enemy superclass that are not listed here, as it is meant to be a flexible class so that we may add and remove different types of enemies at will, as we come up with them.

**4.3 Item Class Outline**



All items in the game belong to the abstract Item class. All of the items can be further subdivided into the Weapon item class and the Consumables item class, both subclasses of the abstract Item class. The Consumables class includes items that are consumed upon use, meaning they can only be used once, and then they disappear. These items are meant to change the player’s statistics, so it will include fields such as the changeHealth, changeArmor, and changeSpeed fields, along with respective getter and setter functions. On the other hand, the Weapon class controls the weapon the player uses, and therefore includes fields such as damage and attackSpeed. The Weapon class can be further subdivided into the SwordWeapon subclass and the BowWeapon subclass, with fields containing animations unique to each class.

**4.4 General Game Flow Outline**



We begin the general game flow outline at the start level position. From there, we move into the wave section of the game. The game spawns a certain number of enemies and then begins a timer. The timer signals when to spawn a new wave. A new wave will either spawn if the timer ends or all the enemies are defeated on the level. This cycle will continue until the player dies. When the player dies, an experience menu will open up allowing them to upgrade their character. After the user selects which stats to upgrade, the game then offers a choice between one of two perks which will greatly impact the player. After the experience points have been allocated and the perk has been chosen, the game moves into a cut-scene which will then prompt the player to make a decision. Once the decision has been made, a new level loads and the process repeats.