David Bond
Matthew Yengle
Timothy Canipe
Vishal Bhatt
Group 6
CSC 490

Project Report 6

Project Name

Dungeon of the Mighty Titan - Vanquish (DMT-V)

Project Goal

To create a multiplayer video game that will allow 3 to 5 players to fight AI controlled monsters in a dungeon-like setting.

Project Description

Over the years, there has been a surge in popularity of massively multiplayer online games. One prominent feature of these games is the dungeon raiding mechanics they employ. Players form groups to fight hordes of enemies and powerful bosses as they travel through dungeons in order to obtain rare items. However, players will usually have to play for many hours before they can experience these adventures, as they require high-leveled and well-equipped characters to survive. With our game, players will be able to form a group and raid dungeons right from the beginning. Furthermore, our game will capitalize on the popularity of several other genres, attracting many video game players.

Note: For video game genre definitions, please see the document on our github: DMT-V/Documentation/Analysis of Video Game Genres.

Subproblems

Beyond scope:

1. Port to iOS, android, linux, etc.
2. Single-player campaign (or balance for single-player play).
3. Aesthetically pleasing - as far as graphics, animations, models, and meshes.

Requirements (UPDATED)

The game will:

1. (Functional) Include multiplayer functionality:
    a. Using a client\Server Model.
    b. An in game chat system will not be provided.
    c. An area for people to find other people to play with will not be provided in the actual game.
    d. Internet connection (56Kbps or faster).
    e. One user designated as the server; all others will be clients to this server
    f. The server (and only the server) will maintain a single copy of the GameMode actor.
    g. The server will provide each client with a GameState actor.
    h. Server will provide clients with current map when they connect.
    i. Successful client connections will receive a temporary PlayerController to replicate.
    j. Remote Procedure Call (RPC) will be used to update map and state changes.
2. (Functional) Allow the player to interact with aspects of the game, such as:
    a. Other player characters in the game.
    b. Enemy units and other non-player characters in the game world.
    c. The game world that the player character is in.
3. (Functional) Save the state of the game.

    a. Any one of the users playing the game should be able to save the state of the entire game.
4. (Functional) Allow a user to load the saved game.
5. (Functional) Have an inventory system with an item database.
    a. Each player character will have their own inventory system.
    b. Players will be allowed to trade items.
    c. Players can add items to their inventory from the game world.
    d. Players can drop items from their inventory in the game world.
    e. Players can equip equipable items.
6. (Functional) A combat system that gives the player options depending on their class.
    a. (Performance) As combat is a major component of dungeon-crawlers, the system should be refined and appealing.
7. (Functional) A turn-based movement and combat system.
    a. (Performance) While retaining fast-paced and exciting gameplay.
8. (Functional) Feature an attribute system.
    a. (Performance) The attribute system will give the user several options on how they want to play the game.
9. (Functional) Give playable character classes unique skills that distinguish them from other character classes.
    a. (Performance) Have enough skills that players will have a choice in how they build their character class.
10. (User Interface) Be represented using a visual method of design (GUI) that the player will be able to view and interact with.
    a. Include a Heads-Up Display (HUD) that the user will be able to see their player character's stats and statuses.
    b. (Performance) Responsive and well-designed layout of the visual design.
11. (System) Be able to run on desktop computers running Windows 7 or newer.
    a. Requires no special hardware; uses standard keyboard and mouse setup.

12. (System) Use Unreal Engine's C++ API and scripting language Blueprint.
13. (System) Development environment will be Unreal Editor and Microsoft Visual C++
14. (DROPPED) Prevent the user from switching characters between instances of different games.
    a. Characters should be locked into one game, from which they cannot be moved.
15. (DROPPED) Prevent the user from tampering with save files in order to change their game in unintended ways.
16. (Functional/Security) Use the Steam SDK package to allow Steam users to join game sessions.

Risk Assessment

1. No member on the team has created a full game before. Not only will it be challenging learning how to make a game, but we will also not be able to reasonably predict what we will be able to accomplish in the given time.
    a. Learning more about video game development from the beginning will help solve problems that may arise due to our lack of experience.

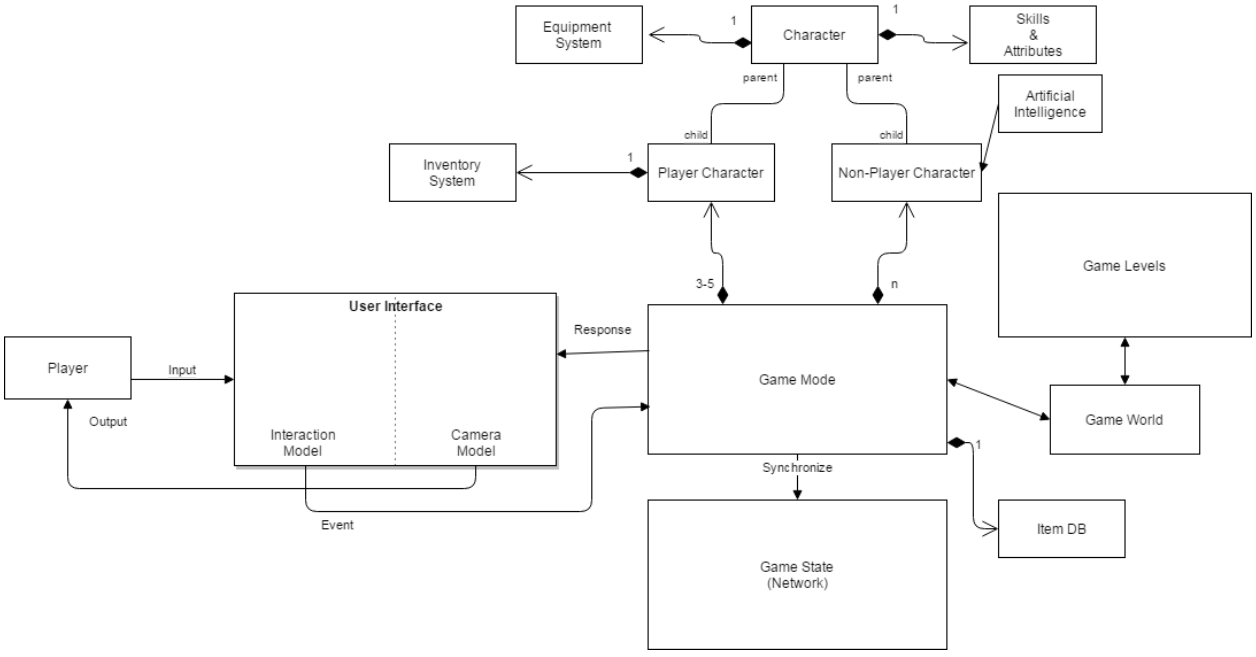
2. The Unreal Engine uses a combination of C++ and Blueprint, the latter being its own scripting language. Not all members are experienced with C++, and no members is experienced with Blueprint.
    a. One member is well-experienced with C++, and there are plenty of available resources for C++.
    b. Epic Games (creators of Unreal Engine) provide many tutorials and guides for Blueprint. C++ can be used for the majority of development in the unreal editor, allowing Blueprint to be learned over time.
3. Video games change quite frequently during development. Using the waterfall method of software engineering may be much less effective in game design.

a. By leaving out a concrete story, we are given many options with the game's development. Furthermore, using a very popular setting (medieval/fantasy) gives us a great deal of available resources.

4. Video game development have many different roles (artists, storyboard writers, animators, etc) that are not fully present in our team full of programmers.
    a. Independent videogame development is a well-supported community on the Internet, with plenty of information, resources, and free assets.

System model:

(Note: Full-sized model will be included in the submitted folder as well as on our github page.)



Subsystems

Game Mode

Description: The Game Mode subsystem will be used as the central hub for the interaction between all other subsystems. Formally, the Game Mode will define all rules for the game. It will setup the start of the game, define the winning conditions and the losing conditions, control the turn-based gameplay, and many other aspects of the

actual game. To control these aspects, the Game Mode class will have instances of the 3 to 5 Player Character classes along with the number of Non-Player Character classes current on the level (denoted in the image as n). The Game World class will also be used to cycle through the different Game Levels, which are changed according to some requirement rules defined by the game mode. Every action performed by the player character should be checked by a rule in the Game Mode class for validation. This will, in turn, allow the Game Mode class to tell the Game State class the changes caused by the player's action. As the Game Mode class does not actually store data on the current game, it will not need to be synchronized between the players of an online game.

Game State

Description: The Game State refers to the complete set of actors and their associated variables. In a multiplayer environment using the client-server model, each actor maintains a list of properties that can be marked for replication to clients. Whenever the value of the variable changes on the server side, the server sends the client the updated value. The variable may have changed on the client, but it will be overwritten by the new value from the server. Property updates only come from the server: the client will never send property updates to the server. Some properties replicate by default (e.g., Location). Replication is reliable, which means that the property of the client version of the Actor will eventually reflect the value on the server(3).

User Interface

Description: The User Interface subsystem will allow the player to interact with the game. In the game, the User Interface will be represented visually by a Heads-Up Display (HUD) showing the player character's status, an inventory and equipment layout, and a menu system for control over the game settings and options. The User Interface will alert the Game Mode when the user performs an event on the interface. In return, it will be expecting some type of response from the Game Mode. The UI will then be updated to relay the response to the user in an understandable form. Not all

responses in the User Interface are a result of the user performing an action. In the Unreal Engine, the user interface will be made using the Unreal Motion Graphics UI Designer.

Game World

Description: The game world will consist of a grid based gameplay. The map(s) that are created will give each player the opportunity to move a certain amount of steps. Having the grid will enable the player to select many different movements.

Game Level

Description: Levels will have subsections where the environment will change after a certain point in time. Having this change will keep the player interested for longer. Essentially the player will start off in a dungeon making their way up to the boss that will be located at the top of the castle.

Character

Description: The Character subsystem will be used as the base class for both the Player Character and the Non-Player Character classes. It also interacts with the equipment and skills & attributes classes.

Player Character

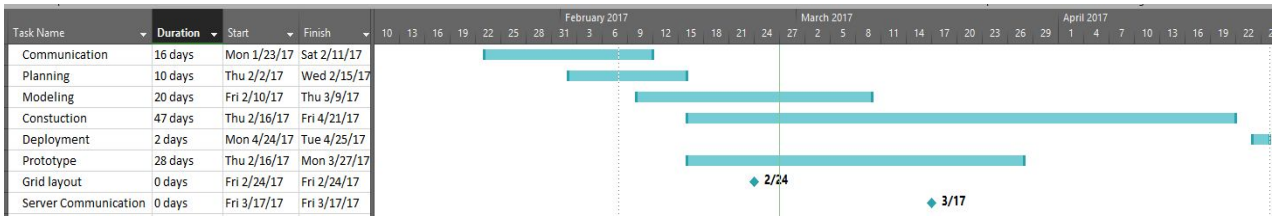
Description: The Player Character subsystem will contain the graphical representation of the character that the user plays as and also communicates with the inventory system, and the Character and Game Mode subsystems. In this game there will be an instance of this subsystem for each of the users who are in the Game World. The user does not interact with the Player Character directly; instead the user interacts with the UI, which interacts with the Game Mode, which then relays the information to the Player Character.

Non-Player Character

Description: The Non-Player Character subsystem will contain the graphical representation of all the enemies within the game. It will also interact with the AI system, and the Character and Game Mode subsystems. In the game there will be an instance of this subsystem for each enemy within the game.

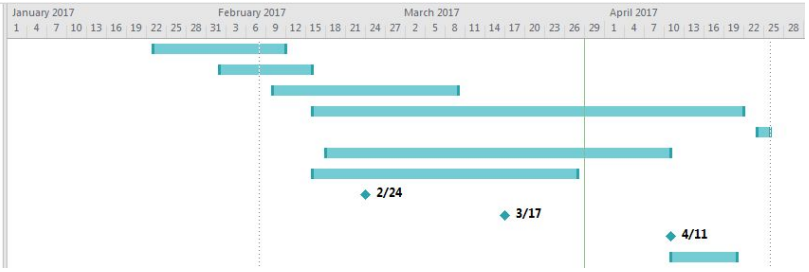
Timeline

(Note: Full-sized Gantt chart will be on our github page).



Task Name	Duration	Start	Finish
Communication	16 days	Mon 1/23/17	Sat 2/11/17
Planning	10 days	Thu 2/2/17	Wed 2/15/17
Modeling	20 days	Fri 2/10/17	Thu 3/9/17
Constuction	47 days	Thu 2/16/17	Fri 4/21/17
Deployment	2 days	Mon 4/24/17	Tue 4/25/17
Prototype	28 days	Thu 2/16/17	Mon 3/27/17
Grid layout	0 days	Fri 2/24/17	Fri 2/24/17
Server Communication	0 days	Fri 3/17/17	Fri 3/17/17

(Updated) Gantt Chart



Task Name	Duration	Start	Finish
Communication	16 days	Mon 1/23/17	Sat 2/11/17
Planning	10 days	Thu 2/2/17	Wed 2/15/17
Modeling	20 days	Fri 2/10/17	Thu 3/9/17
Constuction	47 days	Thu 2/16/17	Fri 4/21/17
Deployment	2 days	Mon 4/24/17	Tue 4/25/17
Multiplayer Development	37 days	Sat 2/18/17	Mon 4/10/17
Prototype	28 days	Thu 2/16/17	Mon 3/27/17
Grid layout	0 days	Fri 2/24/17	Fri 2/24/17
Server Communication	0 days	Fri 3/17/17	Fri 3/17/17
Testable Prototype	0 days	Tue 4/11/17	Tue 4/11/17
Testing	8 days	Mon 4/10/17	Thu 4/20/17

At the bottom of the Gantt chart is a sectioned timeline for a prototype. This prototype will have the basic functionality of connecting players and syncing moves. Once the prototype phase has been completed there will be a rendering and touching up phase in which the game will be finalized and completed.

Group Roles (UPDATED)

Decisions and arguments will be resolved reasonably and democratically, keeping in mind the best interest of the project.

David Bond:

- Implementing multiplayer functionalities, which span several components.
- Programming of the following game menus: Main menu, Lobby menu, and character selection menu.
- Research into the networking aspect of the Unreal Engine.
- Coding of the Save and Load systems.

Matthew Yengle:

- Coding of the Game Mode, Item Database, and User Interface major subsystems. As well as necessary functions to the Player Character class for additional systems.
- Coding of the Equipment and Inventory minor subsystems.
- Research into open-source item models, including weaponry and armor.

Timothy Canipe:

- Coding of the Character, Player Character, and Non-Player Character subsystems.
- Coding of the Skills and Attributes minor subsystems.
- Research into open-source character models, animations, and assets.

Vishal Bhatt:

- Coding of the Game World and Game Level major subsystems.
- Research into open-source world meshes, level designs, and audio assets.

Use cases

Name: Attack an enemy non-playable character (NPC)

Actors: The player character (PC) and the NPC.

Goal: To attack the NPC.

Precondition: It must be the player character's turn, they have not fought this turn yet, and the enemy unit is within range of the player character's attack.

Summary: The user wants to attack an enemy unit. They click “Attack” on the GUI, choose their attack, then click on the enemy unit. If all actions were valid, then the user’s character will attack the NPC. The NPC will be properly damaged to reflect the attack.

Steps:

User	System
The user clicks the “Attack” button on their GUI.	The system processes the input and changes the GUI to acknowledge the user input. It then waits for the next user input.
The user clicks on the NPC they wish to attack.	The system validates the target selected by the NPC. It will then check all other preconditions.
The user is notified as to whether the attack was valid or not through the GUI.	The game state changes to reflect the damage performed to the NPC. The user is unable to attack again this turn.

Post-Condition: The health of the enemy is updated and the player character cannot attack again this turn.

Name: Move the player character

Actors: The player character (PC).

Goals: To move the player character to the designated tile.

Preconditions: It must be the player character’s turn, and the designated destination tile is within movement range.

Summary: The user wants to move their character in the game world. The user clicks “Move” on the GUI, then selects where in the game world they want to move. If the location is valid and the player has moves left, the player will move to that location.

Steps:

User	System
The user clicks the “Move” button on their GUI.	The system processes the input and changes the GUI to acknowledge the user input. It then waits for the next user input.
The user clicks on the space they wish to move to.	The system validates the preconditions.
The user sees the character move to the target location.	The game state will update the new location of the player in the game world.

Postconditions: The player’s move count for this turn is decreased by the number of tiles they moved. If this reaches 0, then the player cannot move again until the next turn.

Name: Pickup an item

Actors: The player character (PC), the item.

Goals: To remove the item from the game world and add it to the player’s inventory.

Preconditions: The item must be within reach of the player character, and the player must have space in their item inventory.

Summary: The user wants to pick an item up from the game world and add it to their player’s inventory. The player clicks on the item in the game world. If the player is within reach and there is room in their inventory, the user will be given the option to add it to their inventory.

Steps:

User	System
The user clicks on the item.	The system processes the input and checks the preconditions.
The user agrees to add the item to their inventory.	The system gives the player the option to add the item to their inventory.

The user can now see the item in their player's inventory.	The game state changes to show the item has been picked up. The player's inventory is updated to include the item.

Postconditions: The item is no longer on the ground. The player's inventory space has less room in it.

Name: Use an item.

Actors: The player character (PC) and the item.

Goals: To use the item's associated effect.

Preconditions: It must be the player's turn, the player must have the item, and the item must be used correctly.

Summary: The user wants to use an item that they have in their player character's inventory. The item's effect should be given to the user in a short description shown when hovered over that item with the mouse actor. The user will double click the item in their inventory with the mouse, and expect the item effect to activate.

Steps:

User	System
The user double clicks on the visually represented item in their inventory UI.	The system finds the item in the item DB, and obtains its associated use.
The user will be able to tell in some way that the item was or was not successfully used.	If the user used the item in a valid method, the associated use will occur.

Postconditions: The item may or may not be destroyed.

Name: Acquire a skill.

Actors: The player character (PC).

Goals: The player uses skill points to acquire a skill.

Preconditions: It must be the player's turn, the player must have enough skill points available, and the player must be a high enough level.
Summary: The player wants to unlock a skill. The player selects the skills button on the UI. If the player has skill points available and is a high enough level, then the user selects which skill to unlock.
Steps:

User	System
The user clicks on the skills button.	The system processes the input and checks the preconditions.
The user selects which skill to acquire.	The system calculates which skills are available.
The user can now use the skill when applicable.	The game state changes to show the reduced amount of skill points available. The skill shows as unlocked.

Postconditions: The player's available skill points decreases by the number of points they used. If this reaches 0, then they cannot unlock any more skills until they obtain more skill points.

Name: Increase an attribute.
Actors: The player character (PC).
Goals: The player uses attribute points to increase their attribute levels.
Preconditions: It must be the player's turn, the player must have points available, and the attribute must not be at max.
Summary: The player wants to increase one of it's attributes. The player selects the stats button on the UI. if the player has points available then the user selects how many points to increase each attribute by, out of it's available points.

Steps:

User	System
The user clicks on the stats button.	The system processes the input and checks the preconditions.
The user selects how many points to use.	The system calculates the total available points.
The user can now see the changes in their stats.	The game state changes to show the reduced amount of points available. The player's stats are updated.

Postconditions: The player's available stat points decreases by the number of points they used. If this reaches 0, then they cannot increase any attributes until they level up.

Name: Save the game.
Actors: The player character (PC).
Goals: The player wants to save the current game state.
Preconditions: The user must be in the game.
Summary: The player wants to save the current state of the game by using a menu system. In Unreal Engine, particular components of the state or the entire game state can be saved using a SaveGame Object. For simplicity, all components of the game state will be saved.
Steps:

User	System
User opens menu system	A menu of options is presented to the player

User selects to Save	Unreal Engine creates a SaveGame Object, which is written to the hard drive. Game play resumes.

Post-conditions: The exact state of the game will be saved to the user's disk drive.

Name: Load the game.
Actors: The player character (PC).
Goals: The player wants to load a previously saved game
Preconditions: The user acting as the server must load the game. The original members of the save file must be connected to that server.
Summary: The player wants to load saved state of the game by using a menu system. In Unreal Engine, particular components of the state or the entire game state can be loaded using a SaveGame Object. For simplicity, all components of the game state will be loaded
Steps:

User	System
User opens menu system	A menu of options is presented to the player
User selects to Load	Unreal Engine loads a SaveGame Object, which is read from the hard drive
User sees previous saved state	Unreal Engine resumes gameplay from the SaveObject state

Post-condition: The users are loaded into game in the exact state as when the game was saved.

Design Choices

i) Use a well-established game engine, or build backend from scratch.

Existing Game Engine (Unreal Engine 4)

+Overall less time spent coding the backend.

+Less risks using an established game engine that is known to work.

+Easier to synchronize different components of the project.

+If we wanted to release this software as open-source, using a well-known engine would allow others to easily modify the game.

-Learning how to use the game engine will take time.

Building Our Own

+More control over how the game will function.

-Problems may arise when attempting to synchronize the game engine with other components.

-Planning other parts of the game will need to wait until the engine is fully mapped out.

ii) Choosing to focus on the gameplay rather than the story.

Gameplay-Driven

+Narrative exposition can be implicitly told through the gameplay (e.g., environment, characters, enemies, etc.).

+In dungeon-crawlers and rogue-like, the mechanics of the game are more important than the story.

-Gameplay can become repetitive and stale; without a story, there will be nothing new after a few hours of gameplay.

Story-Driven

+People may be more interested in the game if it had a compelling story.

+Many resources are available for the medieval fantasy genre.

-Less time will be spent on the mechanics.

-A story can limit our options when designing the game.

iii) Turn-based system instead of free-roam system.

Turn-Based

+Easier to synchronize multiplayer.

+Gives players more time to think about their decisions, which is why turn-based mechanics are often used in strategic games.

+Places more emphasis on planning movements and actions with the entire group rather than individual skill.

+It is easier for new players to get into turn-based games, as they have time to think and react.

+Movement in the game can be synthesized with motion in the gameworld (e.g. flowing river, swarms of non-enemies bats, glowing lights).

-Players may get bored or lose excitement if a teammate is taking too long with their turn. It is for this reasons that the game is limited to a maximum of 5 players per group.

Free Roam

+The combat and movement system would be more exciting, if done properly.

-Planning, resource, and programming intensive for the movement system and other

-Without restricted movements, players can encounter bugs more frequently. With our limited time, we cannot reasonably perform large scale beta testing that is needed to find all possible bugs.

-Balancing will be harder to accomplish if the combat is not limited in some way, which may lead to less skills or attacks being in the game.

Data Dictionary

DungeonGameInstance - Exists only on the server. The GameInstance is called directly after the GameEngine has finished initalizing. The GameInstance is given control and will then initialize other aspects of the system. When finished, the GameMode will be given control of the game. Nearly every class will need a reference to the GameInstance for synchronization across multiplayer.

Field Name	**Data Type**	**Description**
DungeonGameMode	AGameMode	Defines the rules for the game. [1]
DungeonGameState	AGameStateBase	Manages information to be known about for all connected players. Replicated extension of GameMode.[1]
DungeonPlayerControllers	PlayerController[]	An array to hold the controller of all human controlled players connected. The controller persists the entire time the player is connected. [2]
DungeonAIControllers	AIController[]	An array to hold the controller of all AI, non-human controlled players. [3]
DungeonPawns	DefaultPawn[]	An array to hold the pawns for all players (human and AI) in the game world. A one-to-one mapping is created between a Pawn and a PlayerController (or AIController). [4]
DungeonPlayerStates	APlayerState	An array of all the users connected to the server. The object will persist for the entire time the user is connected. [5]
DungeonGameSession	AGameSession	Created and owned by the DungeonGameMode class. Only exists on the server. Acts as an interface for player connections (finding and joining the server). [6]

Algorithm Analysis

Turn Based Algorithms

PlayerQueue

The order of the queue will be decided at the initialization of the GameInstance. The size of the queue will be constant, and the exact length will be dependent on the number of human players (3-5). Therefore, the queue can be implemented as an array of PlayerControllers, with a fixed size. The order of the queue is decided based on each character's Speed stat, where the character with the highest Speed stat is first in the queue. If there is a tie, decide arbitrarily.

Time Complexity: To find the PlayerController for a character (denoted here as N), it will be at N position in the array. Thus, the time complexity of the queue is constant, or O(1).

Space Complexity: Each player character will need to be stored in the queue. For N players in the game, the queue will hold instances of those N players. Thus, the space complexity of the queue is the number of player characters (denoted here as N) in the server, or O(N).

AIQueue

The order of the queue will be decided at the loading of each level, when the AI controlled monsters for that level are created. Although the amount of monsters will vary depending on the level, the number of monsters on a single level will not change. Therefore, the queue can be implemented as an array, which stores the AIController for each AI-driven character. Since the stats of each type of monster

will not be randomized, the queue order can be predetermined based on monster types in the queue. The order between same monster types can be chosen arbitrarily.

Time Complexity: To find the AIController for an AI monster (denoted here as N), it will be at N position in the array. Thus, the time complexity of the queue is constant, or O(1).

Space Complexity: Each AI character on a given level will need to be stored in the queue. For N AI characters in the level, the queue will hold instances of those N players. When the level is changed, the array can be destroyed, and a new array can be made. Thus, the space complexity of the queue is the number of AI characters (denoted here as N) on the level, or O(N).

ItemDB

The item database will be a structure used in the unreal engine for databases. It is indexed based and will be accessed by the game state to populate the world with items and the enemy drops.

Time Complexity: To find an item with index *n* it will be located at position *n*. Therefore, the time complexity is constant, or O(1).

Space Complexity: Each item in the game will be stored in the database. So for *n* items to be stored, it will take up *n* spaces. Therefore, the space complexity is O(n).

AI Pathfinding

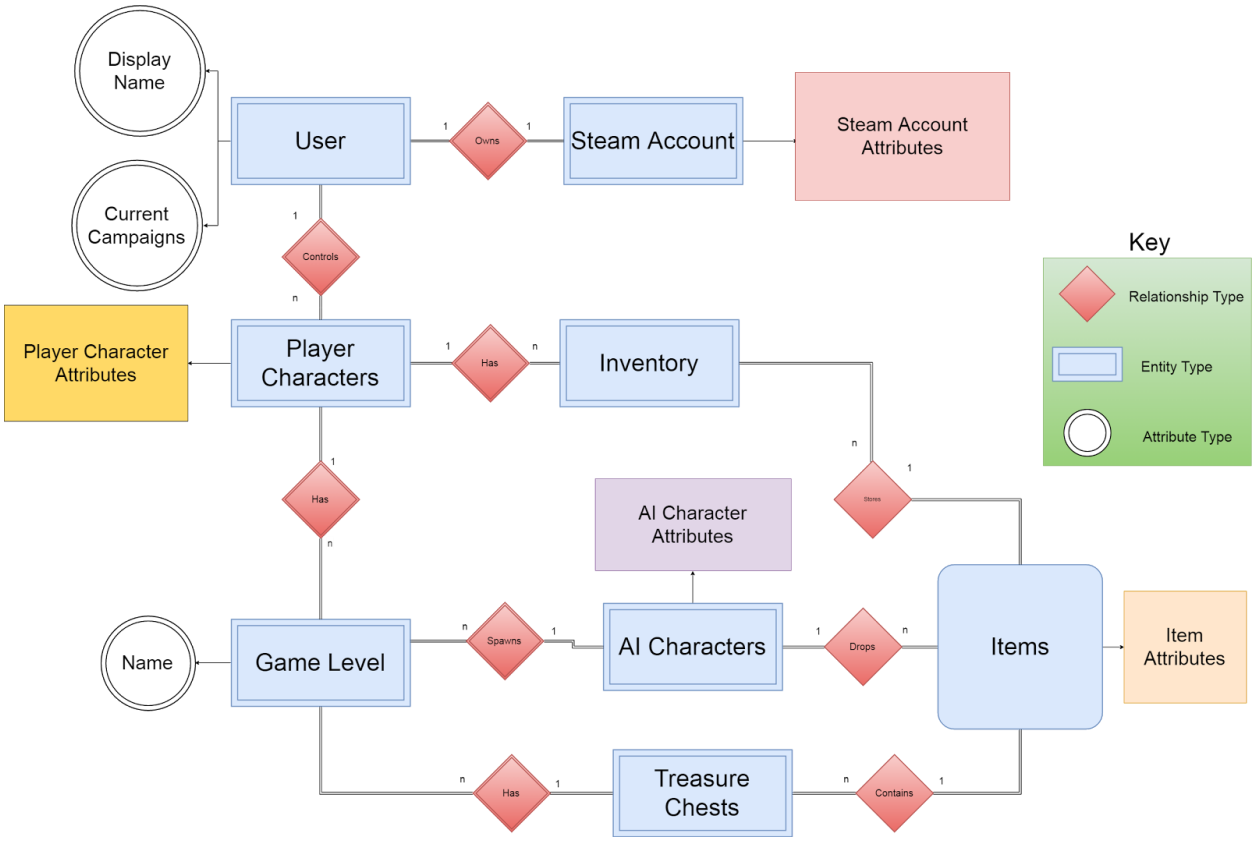
A* is the algorithm that enables AI to find the shortest path to the player. A* is a modified version of the Dijkstra's algorithm. Dijkstra's starts off by calculating the cost of each direction, this algorithm always favors the lower cost. The input is graph that has

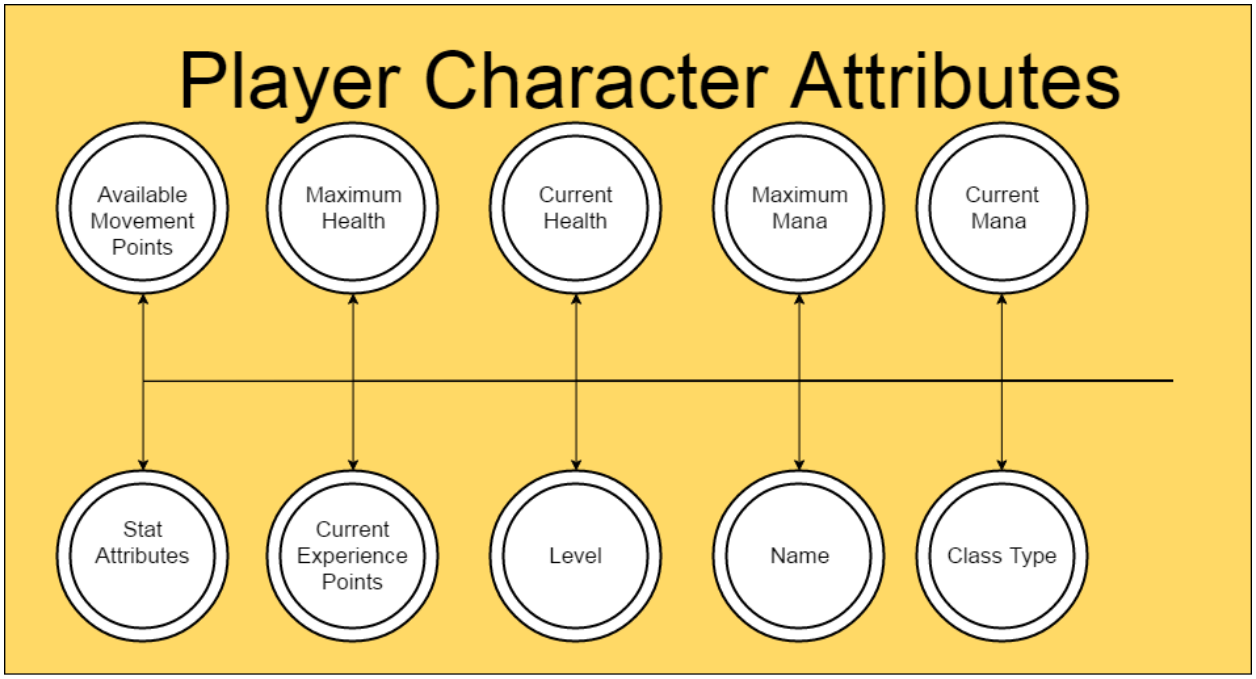
costs it will calculate the cost of each path and will always take the lowest one. A* uses a weighted graph to calculate the lowest cost, much like Dijkstra's, the major difference is that A* calculates the lowest cost of a certain point in the graph - not the entire graph.

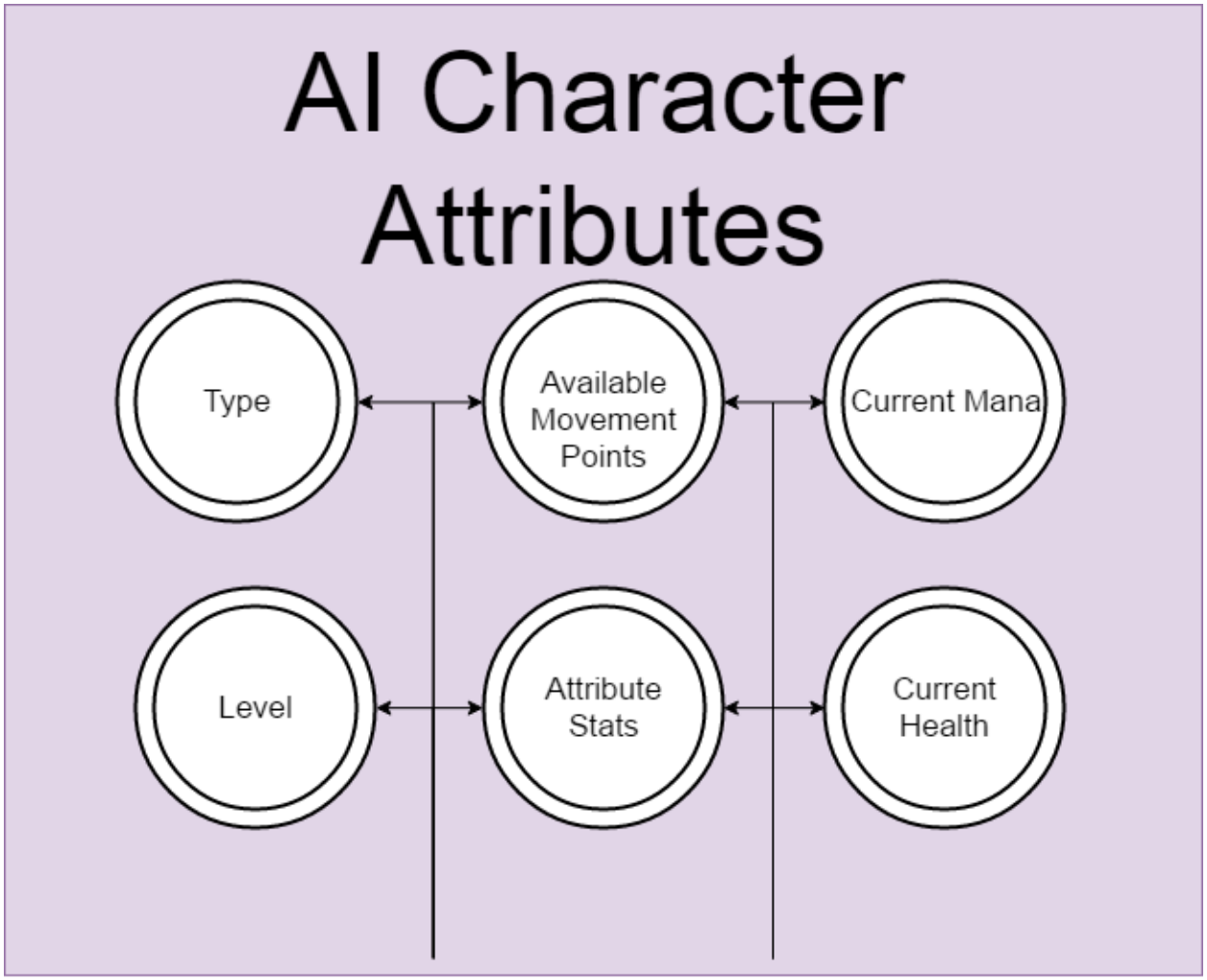
The time complexity of this algorithm is $O(b^d)$ the variable b is the branch of the graph while variable d is the number of nodes creating the shortest path. This algorithm is exponential due to graph having nodes.

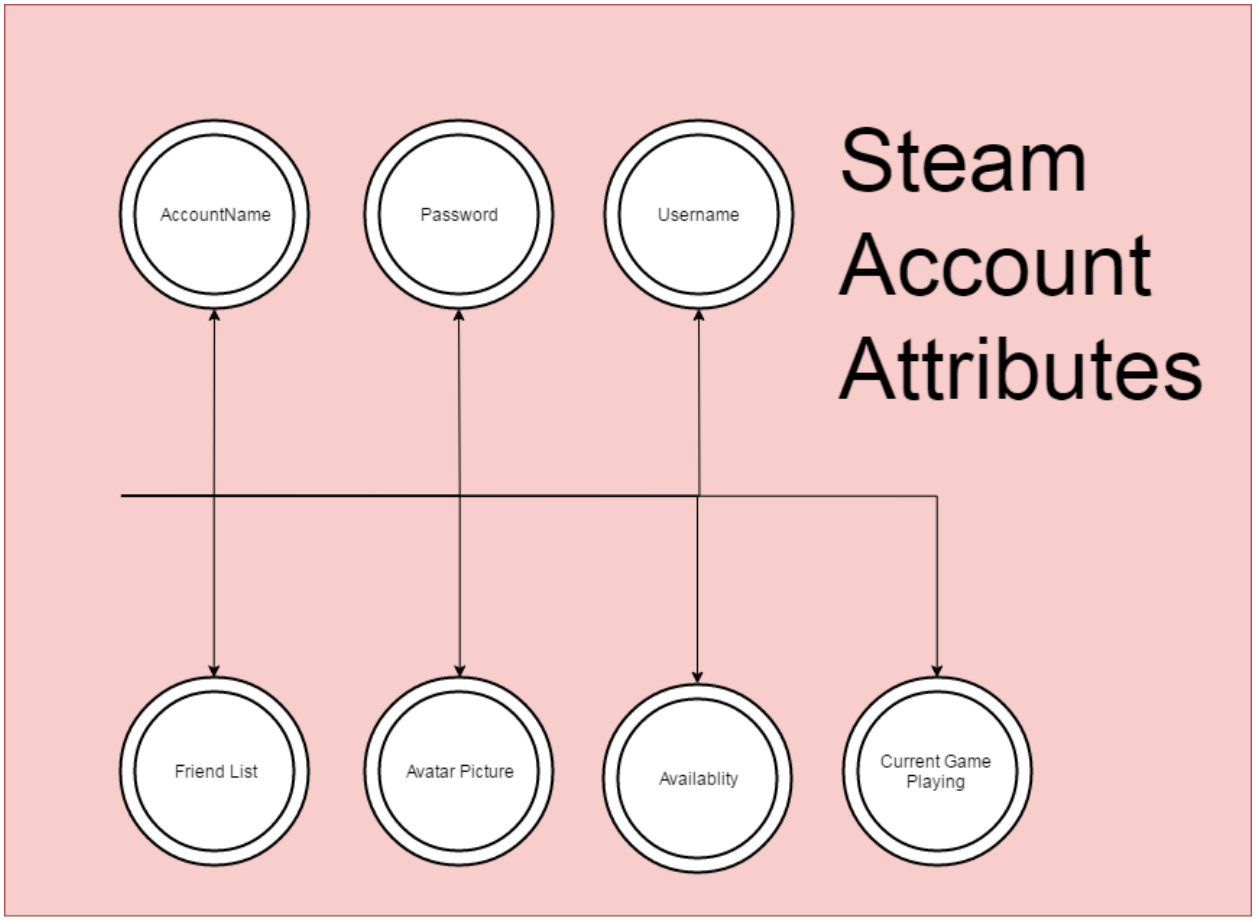
Entity-Relationship

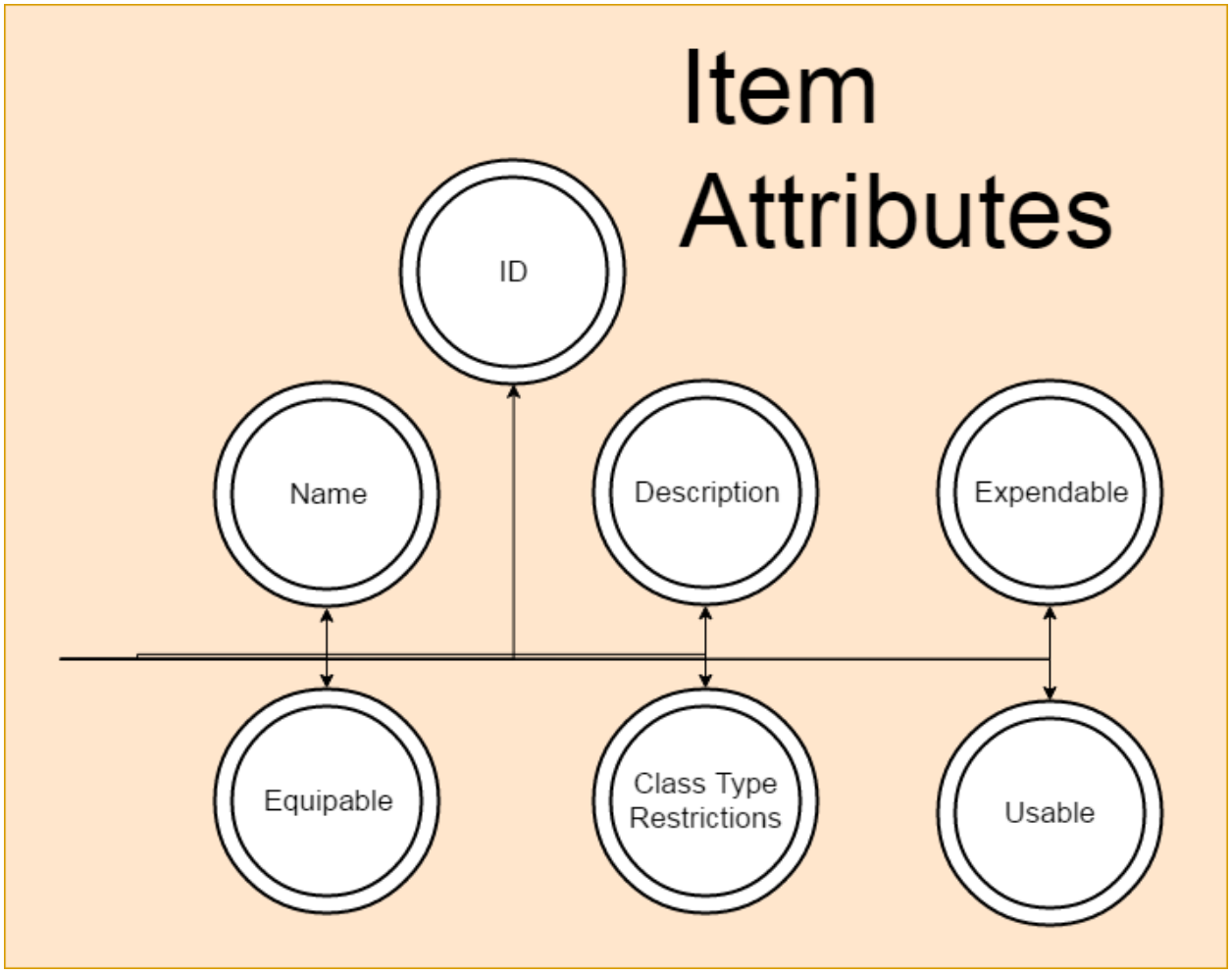
The entity-relationship model shows the relationship between different modules necessary for our game. In turn, each of these modules will have their own attribute types.




Player Character Attributes
Available Movement Points
Maximum Health
Current Health
Maximum Mana
Current Mana
Stat Attributes
Current Experience Points
Level
Name
Class Type


AI Character Attributes
Type
Available Movement Points
Current Mana
Level
Attribute Stats
Current Health

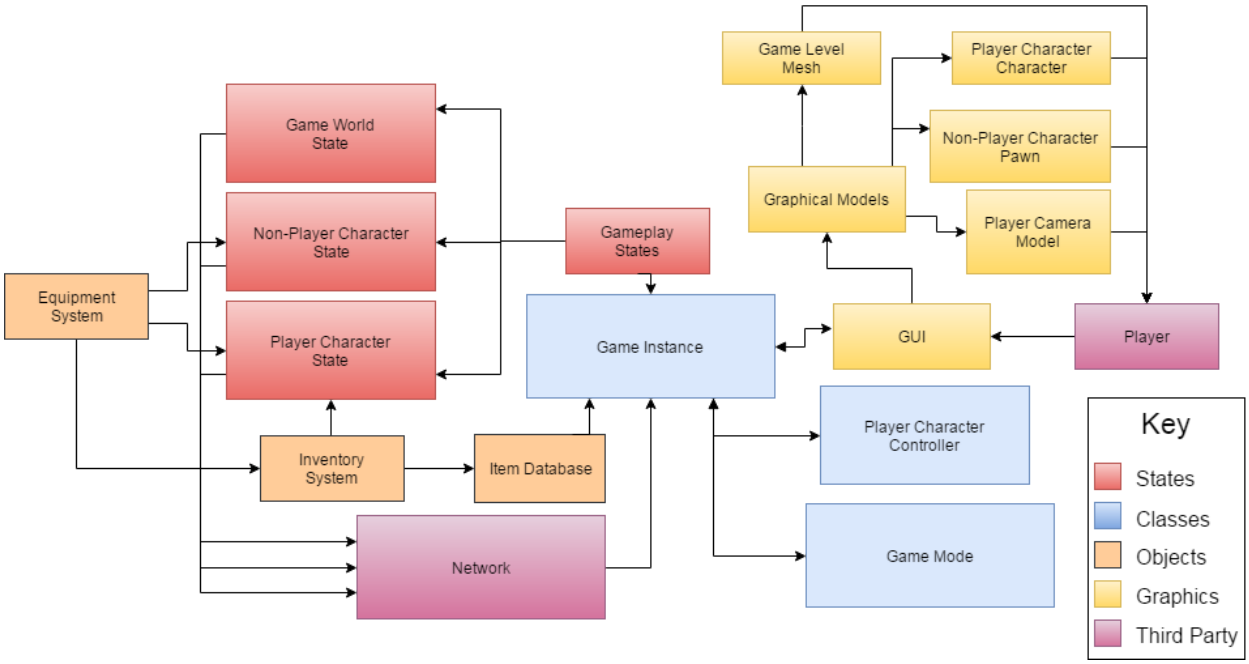

Steam
Account
Attributes
AccountName
Password
Username
Friend List
Avatar Picture
Availablity
Current Game
Playing



Amended Models (UPDATED)

System Model

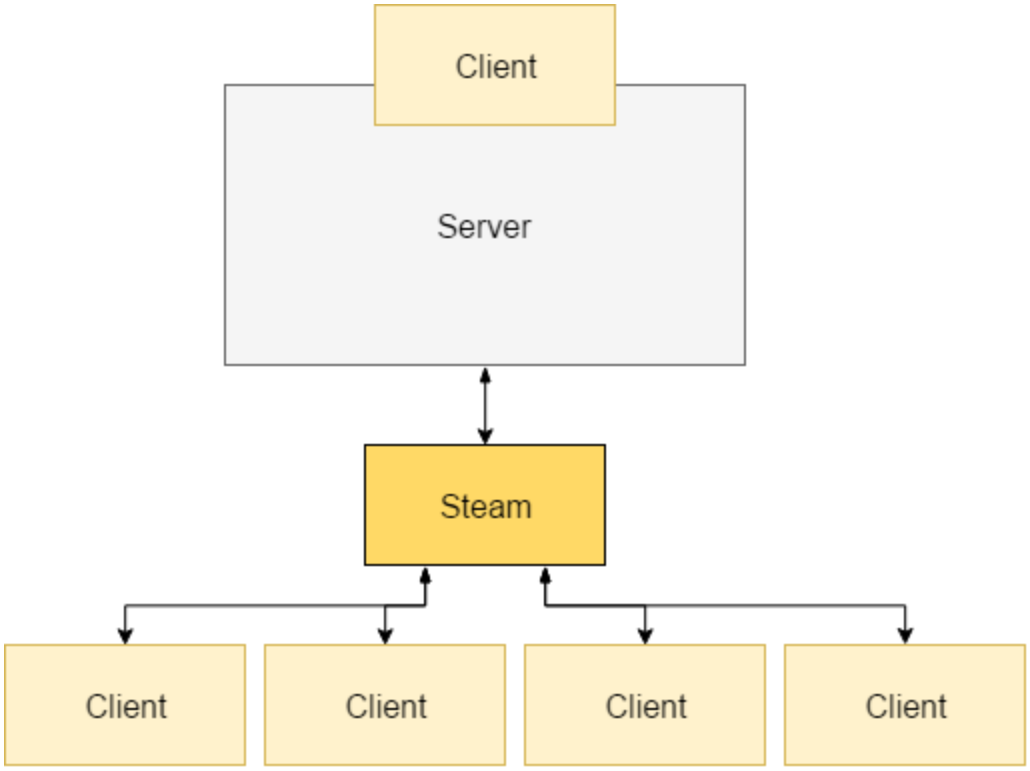
In the previous system model, the Game State component was the central hub of the model. However, the Game State is simply a replicated version of the Game Mode class, which only exists on the server. In other terms, the Game State is a Game Mode class that only the clients interact with, while the server interacts with the Game Mode. Both of these classes may be destroyed and recreated for different stages of the game. On the other hand, the Game Instance class persists in between levels and different stages of the game for the entirety of the game session. This means that all information important to the session should be stored in the Game Instance, such as Player Controllers, GUI models, Game States, etc.



Network Model

Note: For more information on the Steam component, please see the section on Source SDK under the Frameworks header below.

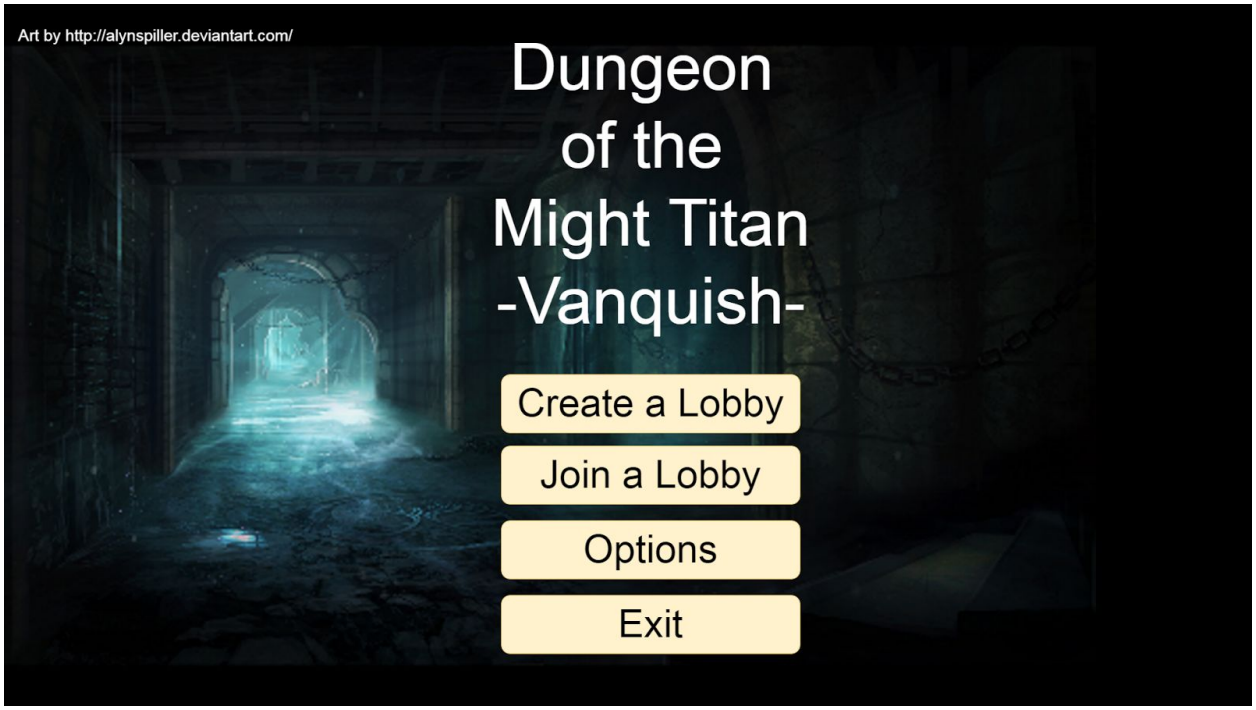
Up to four players will be able to connect to a session over the Internet using Steam. The player that hosts the server will reside on the server itself, and does not have to go through Steam. The players will be able to find sessions that are still in the Lobby by their name. Sessions that are in progress (playing inside of the game world) cannot be joined. Each client has their own PlayerController, and the server holds an additional copy of each player's PlayerController. The server will have the only instance of the GameMode module.



UI Designs (Outdated)

Main Screen

The first interactive screen that the player sees after launching the application.



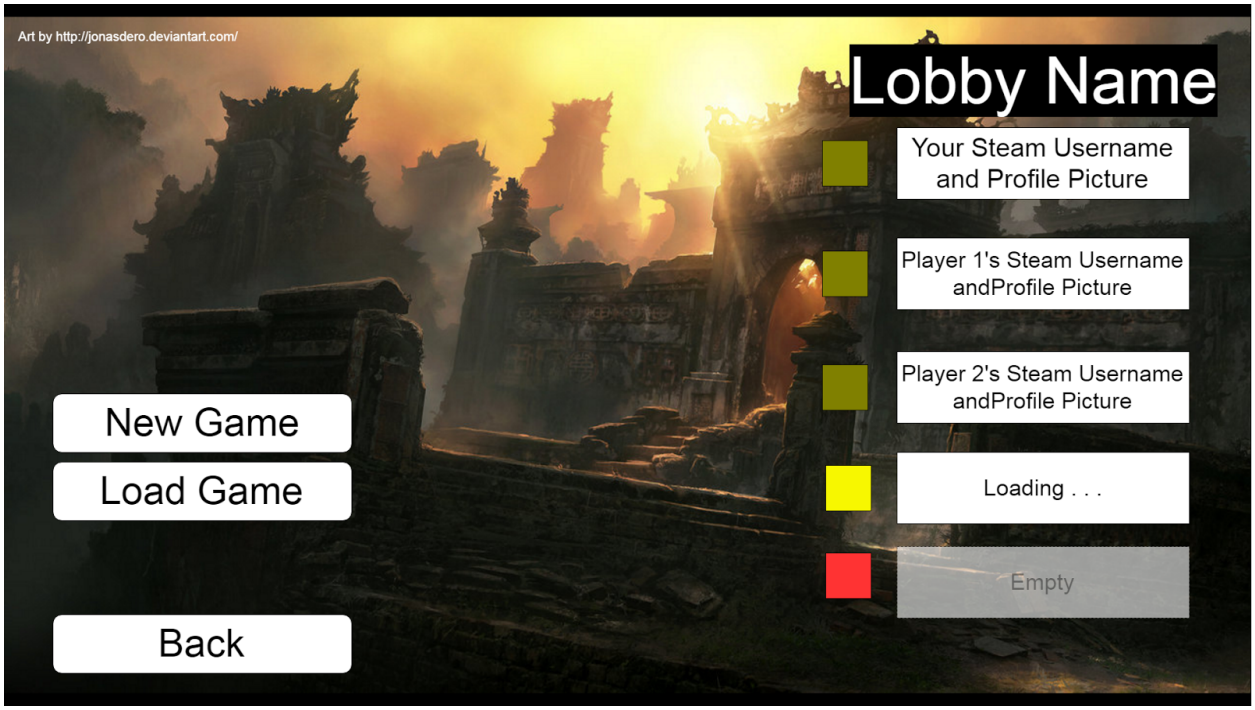
Art by Alyn Spiller [8].

Main Screen Controls

The left mouse button can be used to click on any one of the four buttons displayed on the main screen. Clicking on either “Create a Lobby” or “Join a Lobby” will allow the player to enter information to either create or join the desired lobby. The “Options” button will bring up a list of options that the player can use to change certain game settings, such as the display resolution/ratio, volume controls, video settings, etc. Clicking on “Exit” will close the program.

Game Lobby

After clicking either the “Create a Lobby” or “Join a Lobby” option from the main screen, the player will be placed into a lobby. Only the player that created the lobby will be able to see the “New Game” and “Load Game” options. For each person in the lobby (including the host), their Steam username and Steam display picture will be displayed in a panel on the right side of the screen. A green, yellow, or red box next to their name will show if they are fully loaded into the lobby (green = loaded, yellow = loading, red = empty slot).



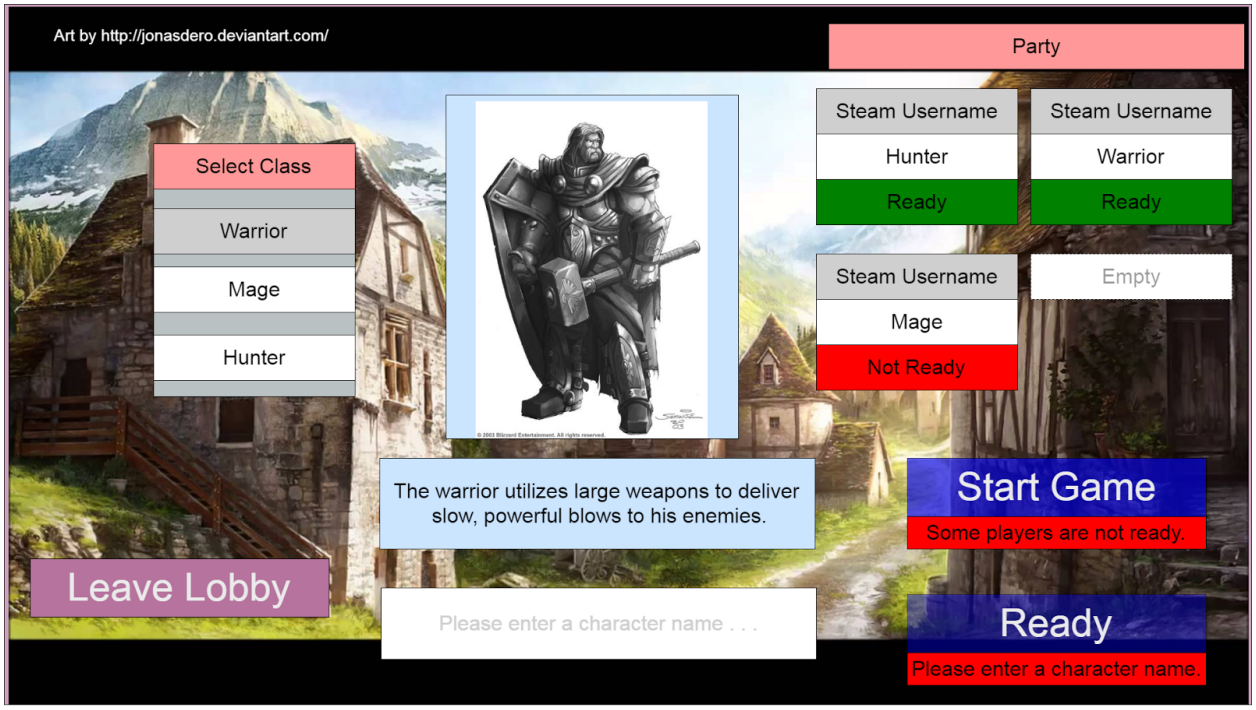
Art by Jonas De Ro [9]

Game Lobby Controls:

For a non-host player in the Lobby, they will only be able to view and interact with the “Back” button. By clicking on this button, they will be returned to the main screen. The lobby host will be able to see all three buttons. If the lobby host clicks on the “Back” button then the lobby will be deleted and all players in the lobby will be placed back on the main screen. The lobby host can only click on the “New Game” or “Load Game” if all player lobbies are either fully loaded in (represented by a green box) or empty (represented by a red box). When clicked, the “New Game” button will bring all players in the lobby to the New Game screen. When the “Load Game” button is clicked, the lobby host will be prompted to pick a Save Game file to load (from their local machine).

New Game Screen

Upon clicking the “New Game” button, all players in the lobby will be taken to the New Game menu to select their character. The design of this interface is still in development, as it appears clunky and cluttered, yet a few more features still need to be added.



Background art by Jonas De Ro [10]

Portrait art by Blizzard Entertainment [11]

New Game Screen Controls:

As with the previous screen, some buttons act differently if pressed by the host vs pressed by a non-host player. If the “Leave Lobby” button is clicked by a non-host, only they will leave the game and be placed at the main screen. If the host clicks on the “Leave Lobby” button, every player in the lobby will be forced to the main screen. Furthermore, only the host player will be able to see the “Start Game” option. This option will only be clickable if all players in the lobby have clicked on the “Ready” button. All players in the lobby can select their character in the box to the left side of the screen. The “Ready” button is unclickable for each party members until they (individually) choose a character and name their character. All players in the lobby can see every other lobby member’s steam username, character selection, and ready status. Once all party members are ready, the host can click on the “Start Game” button and all players will be placed in the game.

Heads-Up Display (HUD) Example

An example of a standard Heads-Up Display, featuring the player's status, play options, current objectives, and the other party members statuses.

Computer game by Square Enix Holdings [12].

Heads-Up Display Controls:

Most of the game can be interacted with by using just the Left Mouse Button (LMB) of the mouse. The bottom left corner will hold a list of interactive command buttons, similar to the image above. However, most of these will only be interactive when it is the player's turn. When it is the player's turn, they can click on the "Attack" button then select an enemy in the game world. If the selected target is valid, then the player will attack that enemy. The player will also have a "Move" button option, that we allow the player to move their character in the game world. When it is their turn, they can click on the "Move" button, then click on a tile in the game world. If the tile is valid, then the player's character will move to that tile. The bottom button will be the "End Turn" button. When a player clicks it during their turn, then their turn will end and move to the next character's turn.

In-game Keyboard Controls:

I - Brings up the player's item inventory.

E - Brings up the player's currently equipped items.

Escape - Brings up the game menu, where the player can adjust settings, save the game to a local disk, or exit the game.

Left Arrow or Right Arrow - Rotates the camera around the viewed player model.

A or D - Switches between the camera models for each player character's pawn.

Frameworks

Advanced Turn Based Tile Toolkit (developed by Knut Overbye):

The Advanced Turn Based Tile Toolkit is a framework that was developed by Mr.Overbye, this framework enables the creators to implement Artificial Intelligence(A.I) and a Grid game play with ease. This toolkit uses the A* algorithm for the A.I, this is the fastest algorithm for searching the shortest path. It offers much more than just A.I it enables the user to have full control and customization of player classes for example; changing the range, the attack, and the search of all characters. Additionally, the toolkit gives a basic grid layout with walls, doors, windows, and much more. The Advance Turn Based Tile Toolkit offers a grid canvas on which the creator can bring their ideas to life.

Source SDK (developed by Valve Corporation):

The Source SDK is a development kit for the Source engine, created by Valve for Steam[7]. Source is used to develop game assets for the Source engine and games, but we will be using it in a limited capacity to support multiplayer. While the Unreal engine does have excellent facilities for multiplayer, they are limited to LAN connections, which doesn't align with our model of multiplayer over an internet connection. Instead, we will be using part of the Source SDK to access the Steam network, which will enable us to negotiate sessions over a WAN with the same ease as with a LAN environment.

Requirements Traceability

COMPLETED - The feature is functionally working as intended; however, it may or may not be in its final state.

PENDING - Progress on the feature has been made, but it is not functioning as intended.

NO PROGRESS - No work has been done to implement this feature.

DROPPED - Features deemed infeasible to implement. Each dropped requirement will have a reason.

1. **(Functional) Include multiplayer functionality:**
    a. **Using a client\Server Model.**
    b. **An in game chat system will not be provided.**
    c. **An area for people to find other people to play with will not be provided in the actual game.**
    d. **Internet connection (56Kbps or faster).**
    e. **One user designated as the server; all others will be clients to this server**
    f. **The server (and only the server) will maintain a single copy of the GameMode actor.**
    g. **The server will provide each client with a GameState actor.**
    h. **Server will provide clients with current map when they connect.**
    i. **Successful client connections will receive a temporary PlayerController to replicate.**
    j. **Remote Procedure Call (RPC) will be used to update map and state changes.**

Requirement 1 supported by:

GameInfoInstance.uasset

ATBTT_PlayerController.uasset

MainMenu.usasset

ServerMenu.uasset

HostMenu.uasset

LobbyMenu.uasset

LobbyPC.usasset

LobbyGM.uasset

Requirement 1 responsibility:

David Bond

2. **(PENDING) Allow the player to interact with aspects of the game, such as:**
    a. **Other player characters in the game.**
    b. **Enemy units and other non-player characters in the game world.**
    c. **The game world that the player character is in.**

Requirement 2 is supported by:

ATBTT_PlayerController.uasset

ATBTT_GameMode.uasset

BP_GridManager.uasset

Town.umap

Requirement 2 responsibility:

Timothy Canipe

Vishal Bhatt

3. **(NO PROGRESS) Save the state of the game.**
    a. **Any one of the users playing the game should be able to save the state of the entire game.**
4. **(NO PROGRESS) Allow a user to load the saved game.**
5. **(PENDING) Have an inventory system with an item database.**
    a. **Each player character will have their own inventory system.**

b. **Players will be allowed to trade items.**
c. **Players can add items to their inventory from the game world.**
d. **Players can drop items from their inventory in the game world.**
e. **Players can equip equipable items.**

Requirement 5 supported by:

Inventory.uasset

Item_Slot.uasset

Item_Drag.uasset

ItemDB.uasset

Inventory_DragDropOperation.uasset

Requirement 5 responsibility:

Matthew Yengle

6. **(PENDING) A combat system that gives the player options depending on their class.**
    a. **As combat is a major component of dungeon-crawlers, the system should be refined and appealing.**

Requirement 6 supported by:

Unit_Parent.uasset

ATBTT_PlayerController.uasset

ATBTT_AI_Controller.uasset

Requirement 6 responsibility:

Timothy Canipe

7. **(COMPLETED) A turn-based movement and combat system.**
    a. **(Performance) While retaining fast-paced and exciting gameplay.**

Requirement 7 supported by:

ATBTT_GameMode.uasset

ATBTT_PlayerController.uasset

Unit_Parent.uasset

ATBTT_AI_Controller.uasset

Requirement 7 responsibility:

Given by Advanced Turn Based Tile Toolkit (by Knut Overbye)

Damage system by Timothy Canipe.

Performance is reflected in all other systems.

8. **(COMPLETED) Feature an attribute system.**
    a. **(Performance) The attribute system will give the user several options on how they want to play the game.**

Requirement 8 supported by:

Unit_Parent.uasset

Requirement 8 responsibility:

Timothy Canipe

9. **(PENDING) Give playable character classes unique skills that distinguish them from other character classes.**
    a. **Have enough skills that players will have a choice in how they build their character class.**

Requirement 9 supported by:

ATBTT_PlayerController.uasset

Unit_Parent.uasset

Skill_Tree.uasset

Requirement 9 responsibility:

Timothy Canipe

10. **(COMPLETED) Be represented using a visual method of design (GUI) that the player will be able to view and interact with.**

a. **Include a Heads-Up Display (HUD) that the user will be able to see their player character's stats and statuses.**
b. **(PENDING) Responsive and well-designed layout of the visual design.**

Requirement 10 supported by:

HUD.uasset

Inventory.uasset

Requirement 10 responsibility:

Matthew Yengle

11. **(COMPLETED) Be able to run on desktop computers running Windows 7 or newer.**
    a. **Requires no special hardware; uses standard keyboard and mouse setup**
12. **COMPLETED) Use Unreal Engine's C++ API and scripting language Blueprint.(**
13. **(COMPLETED) Development environment will be Unreal Editor and Microsoft Visual C++**
14. **(DROPPED) Prevent the user from switching characters between instances of different games.**
    a. **Characters should be locked into one game, from which they cannot be moved.**

Reason for drop:

With an open-source game, player's will be able to customize their playing experience in any number of ways. If a player truly wanted to move their characters between games, they would simply need to modify the game. Therefore, it is not a good use of our time to implement this feature.

15. **(DROPPED) Prevent the user from tampering with save files in order to change their game in unintended ways.**

    Reason for drop (Same as 14):

    With an open-source game, player's will be able to customize their playing experience in any number of ways. If a player truly wanted to move their characters between games, they would simply need to modify the game. Therefore, it is not a good use of our time to implement this feature.

16. **(PENDING) Use the Steam SDK package to allow Steam users to join game sessions.**

Unit Testing in the Unreal Engine

The Unreal Engine supports several categories of testing, all of which are grouped under the Automation System [13].

Feature Test are being performed as we are designing and implemented various functions into the game. The easiest way to test features are when they are being added to the game. Unreal Editor's Blueprint scripting language does not support unit testing at this time. Therefore, feature tests on previously implemented features should be performed periodically to check if it has been disrupted by a new feature.

Content Stress Tests are performed to measure the resources used by game. This test is used to determine the hardware requirements for players that want to play the game. If some areas perform badly in these tests, they may need to be redesigned to improve playability.

For instance, to test the amount of enemies that can be present in the game level at the same time, we created a number of actors and monitored the performance capabilities. With 100 enemies actively in the game level, performance quality did not

suffer a noticeable amount. At nearly 500 enemies, the game was noticeably slowed down; however, it was still playable. Since we do not plan on having anywhere near 100 enemies playing at the same time, the number of enemies on the level should not be an issue. A similar test was performed for the size of the game level itself. It was found that at 50x50 grid (~2500 tiles) the game began to slow down. The map we are using will not be this large. When multiplayer is added to the game, a test should be performed to monitor how much can be sent using Steam's source SDK.

Boundary and range test are performed to prevent A.I from attacking players that are on the opposite side of a wall. Creating the perfect boundary is also very important, having boundaries that are close to grid tend to have a good effect on the game play. The boundary test is essentially making sure the boundaries take up entire tiles rather than partial parts of it. With The advanced turn based tool kit the range of the player tends to bring up spaces that are inaccessible. The solution to this problem would be having walls that fit entire tiles. This also helps with the range test and prevents A.I from attacking players that are on the other side of the wall.

User Training (New)

Main Gameplay

To move the camera around, use the keys **A,W,S,D** to go left, up, down, or right, respectively.
To pan the camera 90 degrees, use the keys **Q** or **E**.
Players are only allowed to move when it is their turn. This is indicated in the upper left hand corner of the screen. Once it is the player's turn, they can move by clicking with the **Left Mouse Button** on a tile within their allowed movement spaces, which is indicated by blue squares.

The player can simply click on a blue tile, and their pawn will move to that tile. If the player does not want to move at all, they can click on the tile that the pawn is current on. The player can perform their movement process over multiple operations.

The static information shown on the screen at all times is called the Heads-Up Display (HUD). The bottom left shows the player's current HP out of their maximum HP, their current mana out of their maximum Mana, and their current experience out of the required experience to level up. A player level's up by defeating monsters and gaining experience. When they level up, they earn Attribute Points that can be used to raise their pawn's attributes, which is discussed later on.

On the right side of the screen, the player can see information about each of their party members. It is in the same order as their own information, with health on top, then mana, and finally experience. Hovering over these graphics with the mouse will show the exact numbers.

The player is allowed one attack per turn, unless otherwise specified by a skill.
To attack an enemy pawn, the enemy must be in range of the player's pawn. A pawn's range is determined by the type of weapon they are using. Melee weapons generally have a range of 1, where the enemy must be on an adjacent tile to hit. Ranged weapons have a range greater than 1. If the enemy is within range, the tile that the enemy is on be red. The player can then click on that tile using the **Left Mouse Button** to attack that enemy. If the enemy is out of range, but the tile is red, then that means the pawn will be close enough to be able to attack that enemy. An example of this is shown below.

The player's turn ends when they have exhausted their options.

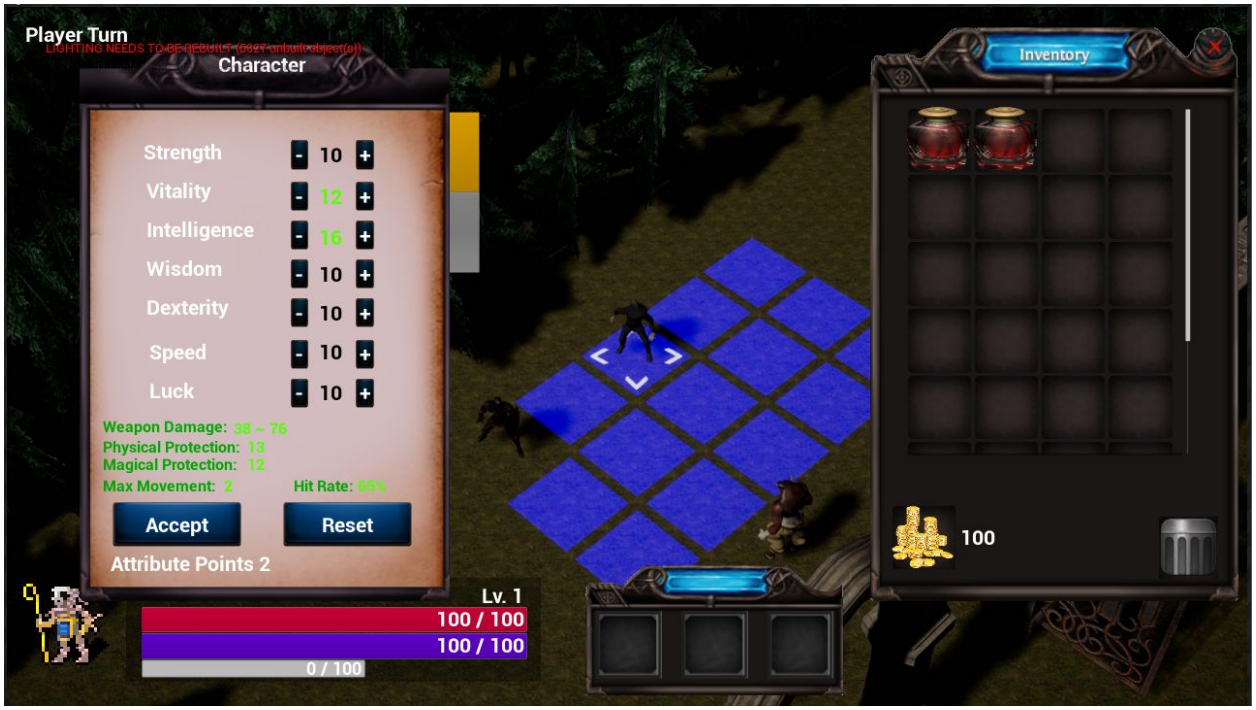
Items are stored in the player's inventory, which can be opened with the key **I**. Nearly all items in the game can be interacted with using **drag and drop** operations. The exception to this are potions, which are consumed by **double clicking** the potion in the inventory. An image of the inventory is shown below, with a few items.

Hover the mouse over most items in the game to display a description of that item. An image of the Steel Dagger's description is shown below.

The player can open the Character menu with the key **C**. The Character menu has two pages of information: the Equipment page and the Attributes page. The Equipment page is shown by default, and is pictured below.

All players will begin a new adventurer with at least one piece of armor and a low-tier weapon. Items can be added or removed to the proper slots by a **drag and drop** operation. On the right, the slots are for: head armor, chest armor, pants armor, and feet armor. On the left are two weapon slots, for such things as daggers, swords, bows, and even shields. Two handed weapons such as bows, staves, and great swords must be equipped in the top left weapon slot. Items can only be equipped from the inventory item.

On the right side of the Character menu, there are two pages that switch between the Equipment page and the Attribute page, the latter is pictured below.



There are 7 attributes in total, which the player can allocate Attribute Points to upgrade. The player earns Attribute Points by leveling up.

- Strength will increase physical, melee damage dealing weapons, such as swords.
- Vitality will increase protection to physical attacks.
- Intelligence will increase magic damage dealing weapons and magic dealing skills.
- Wisdom will increase protection to magical attacks.
- Dexterity will increase the physical, ranged damage dealing weapons.
- Speed will increase the player's Hit Rate (their chance to hit a target) as well as their Maximum Movement spaces per turn.
- Luck will increase the player's chance to land a critical hit, which always doubles the amount of damage the player will deal.

Increasing one of these attributes using the **+** button or decreasing one of these attributes using the **-** button is temporary. The player cancel the changes using the **Reset** button. To finalize the changes, the player must use the **Accept** button. Exiting

out of the menu will cancel the temporary changes. The values shown below the attributes are reflected by any temporary changes.

There are three ways that a player can get items in the game: finding chests in the dungeon, looting defeated enemies, or bought from NPC vendors.
To open a chest, the player must be on an adjacent tile, and it must be their turn. Meeting these conditions, a player can open a chest using the **Right Mouse Button**. A player or enemy cannot walk on a tile that has a chest on it. An image of loot found in a chest is shown below.

The player can drag and drop these items to their inventory; however, they cannot add items to the chest from their inventory.

The second way to get items is by defeating a monster and claiming their loot. A defeating enemy will be replaced by a gravestone upon their death. The same rules that apply to chest concerning how to access it applies to gravestones. However, there are some differences. A player or enemy can walk on a tile with a gravestone on it. If another player or enemy dies on such a tile, their loot will be added to the existing

gravestone. Once the contents of the gravestone are empty, the gravestone will disappear. The items spawned by the death of a low level enemy will be a lower tier than those spawned by the death of a mini-boss or a level-boss. Furthermore, gravestones and chests are the only way to find Gold, which is used as a currency in the game. An image of a chest and a gravestone are shown side by side in the image below. Note that the tile of the chest is not blue, while the tile of the gravestone is blue.

The final way to obtain items is through vendors, which are found throughout the dungeons. Vendors are the only non-enemy NPC in the game, and can be identified by not having a health bar above their head. Player's can access the vendor's merchandise by right clicking on them while it is their turn and their pawn is within two squares of the vendor. Like with chest, neither a player or enemy can walk on a tile with a vendor on it. The player can buy items that the vendor has in their shop by dragging and dropping the item to their inventory. If the player does not have enough gold, they cannot buy the item. The value of the item is shown in its description. The player can also sell items to the vendor. The item must be in the player's inventory, and when

dragged and dropped in the vendor's inventory, the player will automatically receive two-thirds of the item's worth in Gold.

Contribution (New)

Matthew Yengle:

- The following user interfaces: inventory menu, character menu (equipment and attribute page), trade menu, and loot menu (for vendors, chests, and gravestones).
- Item database and all interactions involving items.
- Programming and implementation of gravestone, vendor, and chest.

Resources

[1] Epic Games Inc. (n.d.). *Game Mode and Game State.* [Online]. Available: https://docs.unrealengine.com/latest/INT/Gameplay/Framework/GameMode/, Accessed Mar. 10, 2017.

[2] Epic Games Inc. (n.d.). *PlayerController.* [Online]. Available: https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Controller/PlayerController/, Accessed Mar. 10, 2017.

[3] Epic Games Inc. (n.d.). *AIController.* [Online]. Available: https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Controller/AIController/index.html, Accessed Mar. 10, 2017.

[4] Epic Games Inc. (n.d.). *Pawn.* [Online]. Available: https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Pawn/, Accessed Mar. 10, 2017.

[5] Epic Games Inc. (n.d.). *APlayerState.* [Online]. Available: https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/APlayerState/index.html, Accessed Mar. 10, 2017.

[6] Epic Games Inc. (n.d.). *Sessions and Matchmaking.* [Online]. Available: https://docs.unrealengine.com/latest/INT/Programming/Online/Interfaces/Session/, Accessed Mar. 10, 2017.

[7] Source (game engine), *En.wikipedia.org*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Source_(game_engine)#Source_SDK. [Accessed: 10- Mar- 2017].

[8] A. Spiller, *Dungeon Passage*. 2012.

[9] J. De Ro, *Temple Ruins*. 2012.

[10] J. De Ro, *Oak's Crossing*. 2011.

[11] Blizzard Entertainment, *Paladin Silver Hand*. 2003.

[12] *Kingdom Hearts II.* [Playstation 2 / CD-ROM]. Japan: Square Enix Holdings, 2005.

[13] Epic Games Inc. (n.d). *Automation System Overview*. [Online]. Available: https://docs.unrealengine.com/latest/INT/Programming/Automation/