#1 - Wow...big revisions but Agile allows for that!

#2 - Domain Model - looks good  but is Entity the only Class that has attributes? suspect not. -2 points

#3 - Design Class Diagram - looks good

#4 - ASIM - excellent. Bet that this item was perhaps the easiest for your team to complete.

#5 - Love it!!! Well done.

Excellent job overall!

## Document any additions/deletions/changes made to your Week #1 and/or #2 Deliverable Reports but do not submit any of these again:

After reviewing the requirements and the feasibility for the given timeline, we are removing the following from MVP. These will be features added to the backlog and revisited at a later sprint:

UC4) Deposit RME currency

UC8) Item-shop transaction

R16) (8) The Idle RPG will have an Inventory that displays the Characters Items

R4) ( 6 ) The Idle RPG will allow the user to change the speed the Auto-Pilot makes decisions

R5) ( 8 ) The Idle RPG will provide a minimal graphical interface within the console

R6) ( 6 ) The Idle RPG will include multiple classes for the player to pick from

R7) ( 7 ) The Idle RPG will have different stats and abilities for each class

R8) ( 8 ) The Idle RPG will have items and consumables to help the player character by increasing stats, healing damage, etc…

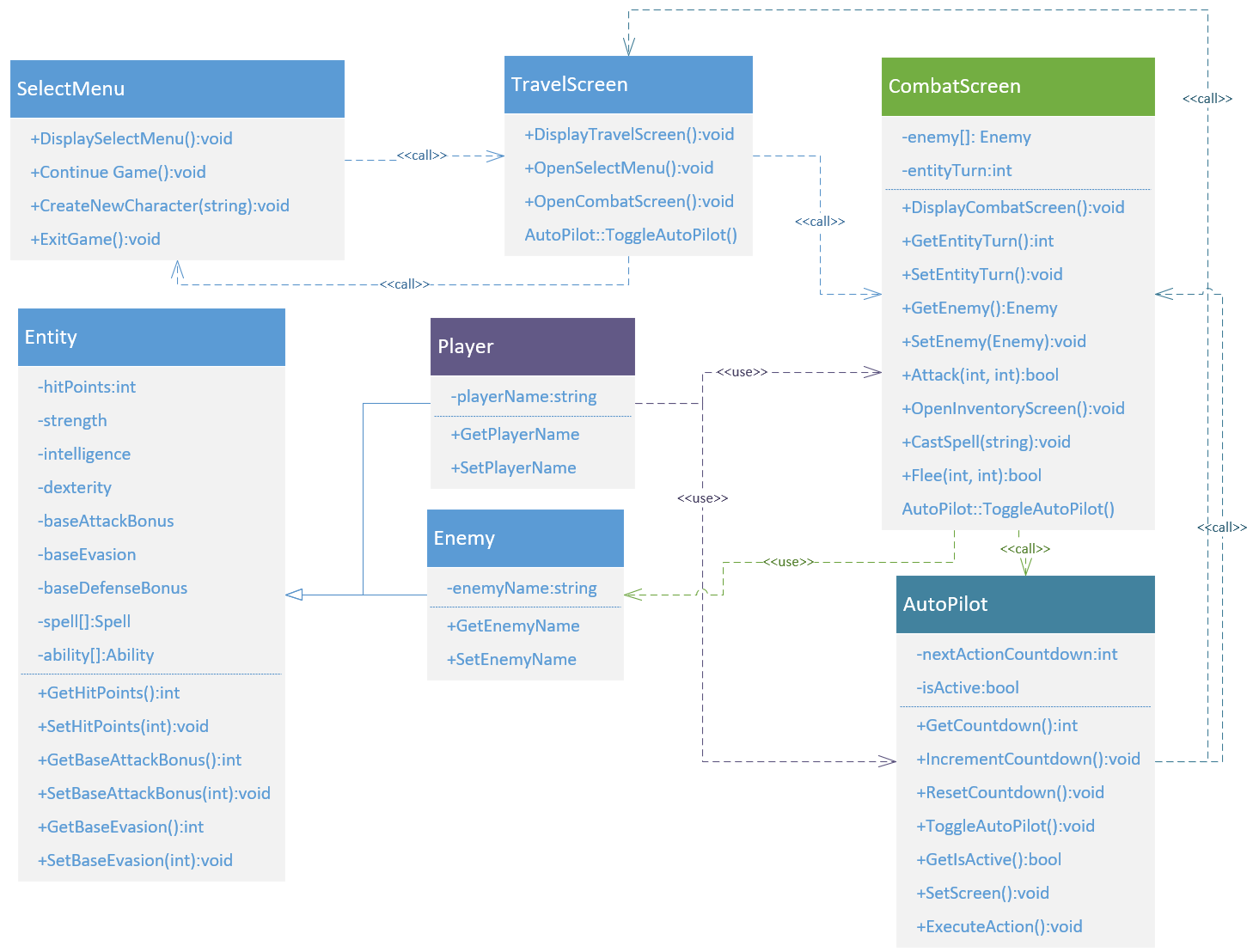
R9) ( 6 ) The Idle RPG will have an in-game currency earned by killing monsters

R10) ( 6 ) The Idle RPG will have a real-money exchange (RME) currency in which players can use real-world currency to buy special in-game currency used for purchasing exclusive items and power-ups

R11) ( 6 ) The Idle RPG will have an item shop for spending in-game currencies and RME on various items and power-ups

R15) ( 5 ) The Idle RPG will provide multiple locations for the player character to move to

## Domain Model of MVP/domain:

1. UML Class Diagram:
2. Actor-System Interaction Model of top 3 Use Cases:

|  |  |
| --- | --- |
| **UC1 - Enable Auto Pilot** | |
| Preconditions: None | |
| Actor: Player | System: IDLE RPG |
|  | 1. System displays autopilot as an option. |
| 1. TUCBW player selects the auto-pilot option | 1. System displays “auto-pilot on” |
| 1. The Player will continue to level up but at a slower rate. | 1. Auto-pilot will disengage automatically when entering the menu screen |
| 1. TUCCW with choosing either Inventory Screen, Menu Screen, or toggling autopilot. |  |
| Postcondition: The System will print “auto-pilot disengaged” | |

|  |  |
| --- | --- |
| **UC2 – Combat Action** | |
| Preconditions- The player will encounter an Enemy | |
| Actor: Player | System: IDLE RPG |
|  | 1. System will present Player with an Enemy along with its stats, weapons, and attributes. Two options will show. 2. Engage in Combat 3. Ignore |
| 1. TUCBW player encounters an Enemy and chooses “Engage in Combat”. | 1. The System will display 2. Attack 3. Item 4. Spell 5. Flee |
| 1. Player will choose Attack. A submenu will show with which weapon to attack with. If Player has one weapon, then it will default to this weapon. | 1. The system will display damage done to Enemy and damage done to Player. 2. If the Enemy is killed a status message will display the Player’s upgraded abilities for slaying an Enemy |
| 1. TUCEW killing an enemy or fleeing an enemy. |  |
| Postconditions: None | |

|  |  |
| --- | --- |
| **UC3 – Create New Character** | |
| Preconditions: The player will start the game for the first time and be prompted to create new character or exit game. | |
| Actor: Player | System: IDLE RPG |
|  | 1. System displays welcome message and instructs Player to create a new character. |
| 1. TUCBW player selects “Create New Character” button | 1. System displays a message to create a new name for the new Character. |
| 1. The player types the desired name and hits enter. | 1. System displays the option to make “new character” (new name) of:    1. Class One or    2. Class Two |
| 1. The Player will select class One or class Two. | 1. The system displays the specific class attributes. The system then displays two options. 2. Choose current class or 3. Go back to class selection |
| 1. The player will confirm class and by selecting choose current class | 1. The system will again, display each attribute with an option of upgrading one attribute. |
| 1. The Player will select upgrade for either 2. Attribute one or 3. Attribute two or 4. Attribute three | 1. System will display the new upgraded attribute, name, and class. 2. ask player to confirm choices made, yes? 3. No? |
| 1. TUCCW Player confirming character choice |  |
| Postconditions: None |  |

1. Sequence Diagram and narrative:

### Narrative:

User selects “Continue Game” from SelectMenu. SelectMenu calls the DisplayTravelScreen() from TravelScreen which send the user a message saying the character is wandering around the forest. This also initiates a timer which after 5000ms calls OpenCombatScreen() from the CombatScreen class. CombatScreen presents the user with a description of the monster and some of its stats along with a menu of options: Attack, Item, Spell, Flee.

The user is now in the combat loop. If the user selects Attack, the CombatScreen will take the return value from Player::GetBaseAttackBonus() and send it as a parameter within the CombatScreen::Attack(int, int) method. Attack() will call Enemy::GetBaseEvasion(). It will then generate a random number (1-100), add that to the first *int* passed in and compare the end result with the Enemy::GetBaseEvasion().

If the number is greater, the Attack() will return true and Enemy::SetHitPoints(int) will subtract the second *int* of attack from the result of Enemy::GetHitPoints(). If the result is greater than zero, Enemy will Attack() Player and the reverse comparison will occur. If the end result is less than or equal to zero, for Player or Enemy, the recipient is defeated, the game breaks the combat loop. CombatScreen then calls DisplayTravelScreen if Player is the victor, or calls TravelScreen::OpenSelectMenu() and presents the user with a message informing them of their untimely death if Enemy won.

If the user selects Flee during the combat loop, CombatScreen::Flee() will generate a random number (1-100) and add that to the first *int* passed as a parameter within CombatScreen::Flee(int, int). Flee() will call Enemy::GetBaseAttackBonus() and compare with the resulting Flee *int*. If greater than Enemy, Flee returns true, Player breaks the combat loop, CombatScreen calls DisplayTravelScreen, and the user is presented with a message informing them they have escaped. If Flee returns false, Enemy will get a free Attack() on Player and then the combat loop will continue to Enemy turn.

Diagram

Description automatically generated with medium confidence