Initial Design

Avita Sharma, Eric Wyss, and David Taus

Team Name: DragonSlayers

Current Group Members:

Avita Sharma, Eric Wyss, and David Taus

Project: DM or Die

A text based adventure game, where the user controls the dungeon master. They can manipulate intelligent 'player characters' who interact in the game. Example: the user can decide if a player succeeds or fails at a task. These players are concurrent processes which make demands on the dungeon master. These demands must be answered in real time or a default behavior is triggered, the default generally having a negative consequence. Multiple demands can come from each 'player character' at the same time, and the user must decide which one to interact with (or if they are quick, try to do them all). The user can directly control monsters and non-playable characters (NPCs) for the 'player characters' to interact with.

Minimum deliverable:

* Split Screen graphical interface.
* Two 'Player Characters' with a set alignment and class.
* One Encounter involving talking to one NPC and one battle.
* The default map has a dungeon and tavern.
* Ability to pause the game.
* Default/unchangeable inventory.

Maximum deliverable:

* Implement up to six characters.
* More classes, more monsters.
* Ability to add more places and characters to the map.
* Add more encounters. Increase the complexity of encounters.
* Basic Excitement meter for each 'player character'. The game is scored on total excitement. Excitement is a metric for how much fun the 'player characters' are having with the user's decisions. (i.e. not purposely failing the battle.) Excitement $\in \mathbb{R}$.
* More user customizable options for the campaign.
* Option to save the game.

First Step:

* One ‘player character’. (One alignment, one class.)
* Message handling between user and ‘player character’.
* One NPC character for the user to control.
* One combat encounter; one monster for the user to control.
* End of combat ends the game with, ‘See you next session!’
* Combat ends if the ‘player character’ dies or the user’s monster (dragon) dies.

Foreseeable Problems:

* Good design documents.
* ‘Player characters’ spam messages too fast for the user to keep up.
* Too slow response times.
* Not enough play-testers.
* Too difficult.
* Handling contradictory events.

Design Decisions

Concurrent Messaging

Solutions:

* Use Threads for our PCs and NPCs, and a global messaging queue to communicate between threads
* Use the multiprocessing library to handle communication between NPCs and PCs, which are separate processes
* Use pyro to handle communication

Chosen Solution:

* We will use threads and a global messaging queue (a monitor), at least for now, since this seems to be the best for local concurrency (not distributed) and the easiest to integrate with the rest of the game. For the queue, we will use a dictionary where each character has its own messaging queue.

Dialogue

Solutions:

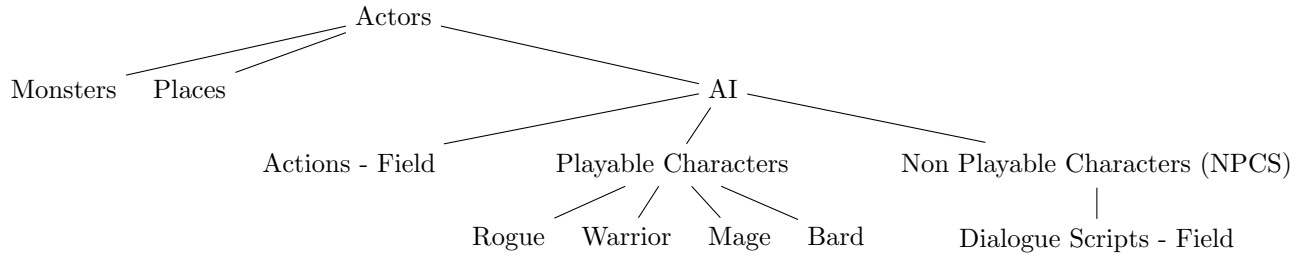
* Use dialogue trees in XML format to represent branching choices
* Use a script-based representation, where each choice calls a different function

Chosen Solution:

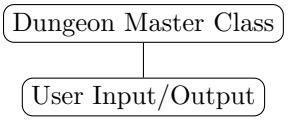
* We will use a script-based representation of dialogue because we have more experience using it and it lends itself well to many small interactions.

Class Diagrams

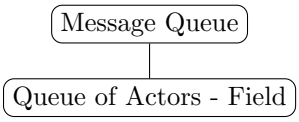
Character Class Hiearchy:



Dungeon Master:



Messaging:



Split Screen GUI:

The Game State will be a tuple containing a dictionary of all the AI and Places in the Game (keys are the name or ID of the Actor, values are the Actor), The Message Queue, and the Game Lock. We may create a Game State Monitor Class to ensure that clients use the lock when accessing or modifying the Game State. The Game State will be passed to all Actors on the Board and the Dungeon Master.

AI are goal-based agents who use utility to decide actions. The current Goal is selected using a transition matrix (Markov Chain). For the current Goal, the action with the maximum expected utility is chosen, and the Actor attempts to perform the action. If a contradiction arises, the Actor chooses a new Goal (it can be the same one). Otherwise, the Actor compares the utility gained (or lost) to their expected utility. If they succeeded, their probability of success of performing that action increases. If they failed, their probability of success of that action decreases. This ensures that the AI is more likely to continue to do the action if they perform successfully, and less likely to do it if they fail.

UML Diagrams

Part 1

```
I decided to Ask
This action has expected utility: 74.7424434927
This action has total utility: 87.7982130336
The Rogue asks The Old Man for money.
The Old Man replies, GET AWAY YOU MONSTER.
My success: 1
I failed by: 0.0

My new success: 0.001
I decided to Make Money
Rogue has 0 zenny

The Old Man has 100 zenny

The Don't go Inn has 0 zenny

This action has expected utility: 60.0
This action has total utility: 100
Rogue has 0 hidden zenny

The Old Man has 0 hidden zenny

The Don't go Inn has 150 hidden zenny

This action has expected utility: 60.0
This action has total utility: 150
Oh no! The Rogue got caught stealing :(

My success: 0.4
I failed by: 0

My new success: 0.001
I decided to Make Money
Rogue has 0 zenny

The Old Man has 100 zenny

The Don't go Inn has 0 zenny

This action has expected utility: 60.0
This action has total utility: 100
Rogue has 0 hidden zenny

The Old Man has 0 hidden zenny

The Don't go Inn has 150 hidden zenny

This action has expected utility: 0.15
This action has total utility: 150
The Rogue pickpocketed The Old Man for 100 zenny!!

My success: 0.6
I succeeded by: 1

My new success: 1.0
I decided to Make Money
Rogue has 100 zenny
```

Part 2

```
The Old Man has 0 zenny

The Don't go Inn has 0 zenny

This action has expected utility: 0.0
This action has total utility: 0
Rogue has 0 hidden zenny

The Old Man has 0 hidden zenny

The Don't go Inn has 150 hidden zenny

This action has expected utility: 0.15
This action has total utility: 150
The Rogue stole from The Don't go Inn for 150 zenny!!

My success: 0.001
I succeeded by: 1

My new success: 0.002
I decided to Ask
This action has expected utility: 0.0346739729979
This action has total utility: 72.2300783778
The Rogue asks The Old Man for money.
The Old Man replies, GET AWAY YOU MONSTER.
My success: 0.001
I failed by: 0.0

My new success: 0.001
I decided to Make Money
Rogue has 250 zenny

The Old Man has 0 zenny

The Don't go Inn has 0 zenny

This action has expected utility: 0.0
This action has total utility: 0
Rogue has 0 hidden zenny

The Old Man has 0 hidden zenny

The Don't go Inn has 0 hidden zenny

This action has expected utility: 0.0
This action has total utility: 0
I went to the dungeon and got eaten by a Troll.
```

The Rogue (Playable Character) is in the 'Make Money' goal state, and decides to ask the Old Man (NPC) for gold. The Old Man doesn't want to give away his gold, so the asking action fails. The Rogue then decides to steal from the Don't Go Inn (Tavern), but is caught stealing. The Rogue decides to pickpocket the Old Man, and succeeds! The Rogue attempts to steal from the Don't Go Inn again, and this time succeeds! The Rogue decides to ask the Old Man for money again, but the Old Man refueses to give any away. Since the Rogue cannot gain any more money from the game board, they decide to go to the dungeon and fight monsters. (Unfortunately, they die.)

For the game, any time a Playable character attempts to do an action requiring a success or fail decision, the user will be able to interrupt the action, and decide if the action succeeds or fails. In the above picture, this is randomized. The above picture also includes utility calculations where we can see how the Rogue decides on what action to do, this will not be outputted for the game.

Table 1: Timeline

11/7	Initial Design.
	Main Loop, GUI, AI, Dungeon Master, Actions, Dialogue, NPCs, Messaging, User Input/Output, Rogue, Monster - Dragon, Minimal Combat, Accomplish as much Minimum Deliverable as possible.
11/21	Refined Design, First Step Done.
	Finish Minimum Deliverable, Add more features–Pause the Game, Save the Game, More Dialogue/Events, More Characters, Releases to Different Systems, Accomplish as much Maximum Deliverable as possible.
12/9	Project Final Report Due.

Development Plan

So far we have made the base AI class, Action class, and Messaging class, the last one is mostly written in pseudocode. By the refined design, we plan to expand the AI class and add the Playable Character and NPC subclasses. We will also integrate messaging into the AIs, create the Dungeon Master class, add a GUI, and create the main loop. We aim to have all of our minimum deliverable done by November 21st. After that, the game will expand by adding in more characters, interactions, battles, and other features. We will also consider releasing the game for multiple devices. Provisionally, the work will be divided as follows:

* Avita - AI, Dungeon Master, and Playable Character classes
* Eric - NPC and Objects classes, GUI interface, and main file
* Taus - Message Handling class, dialogue, and interactions