Arcane Conquest

1. Project Overview

The project will be a roguelike survival game built with Python using Pygame for graphics and event handling and TCOD for advanced tile management and procedural dungeon generation. The game combines the themes of magic and vampire survivor(game) into another fantasy roguelike where the player's goal is to defeat a boss wave every 3 minutes and survive until the 15-minute mark. With four character classes, each will have unique perks aligned with their combat style (magic or weapon-based)

2. Project Review

The game introduces a combined mechanics by blending elements from "20 Minutes Till Dawn" and "Vampire Survivor". The core combat system revolves around mastery of both magic and weapons. There will be three primary classes and a special class, each with distinct playstyles and abilities:

1. **Mage:** This class relies solely on magic as their primary arsenal. Mages can cast powerful spells with high area-of-effect damage, providing strategic advantages in crowd control and boss fights.
2. **Martial Artist:** Focused on weapon mastery, the Martial Artist can wield a variety of weapons ranging from ranged to close combat. While weapons deal more damage per minute and have higher damage rates, their area-of-effect is less compared to magic. This class emphasizes precision and tactical positioning.
3. **Conjurer:** The Conjurer is a hybrid class that can utilize both magic and weapons. However, their level cap for both magic and weapon upgrades is reduced, offering more flexibility but requiring players to

balance between their dual abilities. This class excels in adaptability and strategic versatility.

4. **Vampire (Special Class):** Vampires primarily use blood magic, sacrificing their own health to deal significant damage. Their abilities output the highest damage among all classes and include lifesteal, allowing them to replenish health by attacking enemies. This high-risk, high-reward mechanic provides a thrilling gameplay experience.

Planned Enhancements:

The project aims to build upon the foundational class system by introducing a dynamic Awakening mechanic. Upon reaching level 15, each class undergoes an "awakening," significantly enhancing their core abilities and granting powerful upgrades in combat capabilities. These upgrades will improve various aspects of the classes, such as:

Increased damage output: For example, a 20% increase in damage for weapons or magic.

Expanded area-of-effect (AoE): Spells or weapon attacks will cover a larger area, making it easier to engage multiple enemies.

Enhanced projectile quantity or range: In the case of magic, the number of projectiles or their range will be increased, adding more versatility in combat.

This Awakening system introduces a clear progression and rewarding experience, where the strategic choice between magic or weapons becomes more impactful as players advance.

https://www.youtube.com/watch?v=S_NQmXUC1Sc

(Vampire Survivors Gameplay)

https://www.youtube.com/watch?v=HL8jo7bqRg0

(20 Minute till dawn Gamplay)

3. Programming Development

3.1 Game Concept

Game Mechanics & Objectives:

Core Gameplay: Players select one of four characters, each with distinct abilities, strengths, and perks that align with either magic-based or weapon-based combat styles. Once the game starts, players navigate a large, potentially endless map, facing increasingly challenging enemy waves.

Time-Critical Survival: Every 3 minutes, a boss wave will spawn. The game is designed for a 15-minute survival challenge. Each boss wave tests the players' strategic decisions, resource management, and mastery over character-specific skills.

Character Progression: As players defeat enemies and overcome boss waves, they accumulate points to unlock temporary power-ups and permanent character perks, enhancing replayability.

Items Spawning: Various items such as power-ups, health potions, and weapon upgrades will spawn throughout the map, providing strategic advantages and encouraging exploration.

Key Features:

Varied Character Classes: Four classes with unique traits

Merge of Magic and Weapon Combat: Allowing for strategic depth and diversification in combat.

Items Spawning: Adding a layer of exploration and strategic resource management.

3.2 Object-Oriented Programming Implementation

1. GameManager

Role: Central controller for game flow, state management, event loop, and overall time/wave management.

Key Attributes:

- current_wave
- time_elapsed
- wave_timer
- score

Key Methods:

- start_game()
- update()
- handle_events()
- spawn_boss_wave()
- clear_boss_wave()
- end_game()

2. Player

Role: Represents the user-controlled character.

Key Attributes:

- health
- mana
- position
- class_type (e.g., magic or weapon)
- perks

Key Methods:

- move(direction)
- attack()
- take_damage(amount)
- use_ability()
- update()

3. AbstractEnemy

Role: Basic adversaries that challenge the player in each wave. Acts as a base class for shared attributes and methods of enemies and bosses.

Key Attributes:

- health
- position
- speed
- enemy_type

Key Methods:

- move()
- attack()
- detect_player()
- take_damage(amount)

4.Enemy(AbstractEnemy)

Role: Basic adversaries that challenge the player in each wave.

Key Attributes:

- (Inherit all attributes from AbstractEnemy)
- exp_drop_rate

Key Methods:

- (Inherit all methods from AbstractEnemy)

5. Boss(AbstractEnemy)

Role: A high-difficulty enemy spawning every 3 minutes with unique abilities.

Key Attributes:

- (Inherit all attributes from AbstractEnemy)
- exp_drop_rate
- special_ability
- special_loot_drop

Key Methods:

- (Inherit all Methods from AbstractEnemy)
- use_special()
- update()
- drop_special_loot()

6. Item

Role: Represents in-game collectibles such as power-ups, health potions, and weapon upgrades.

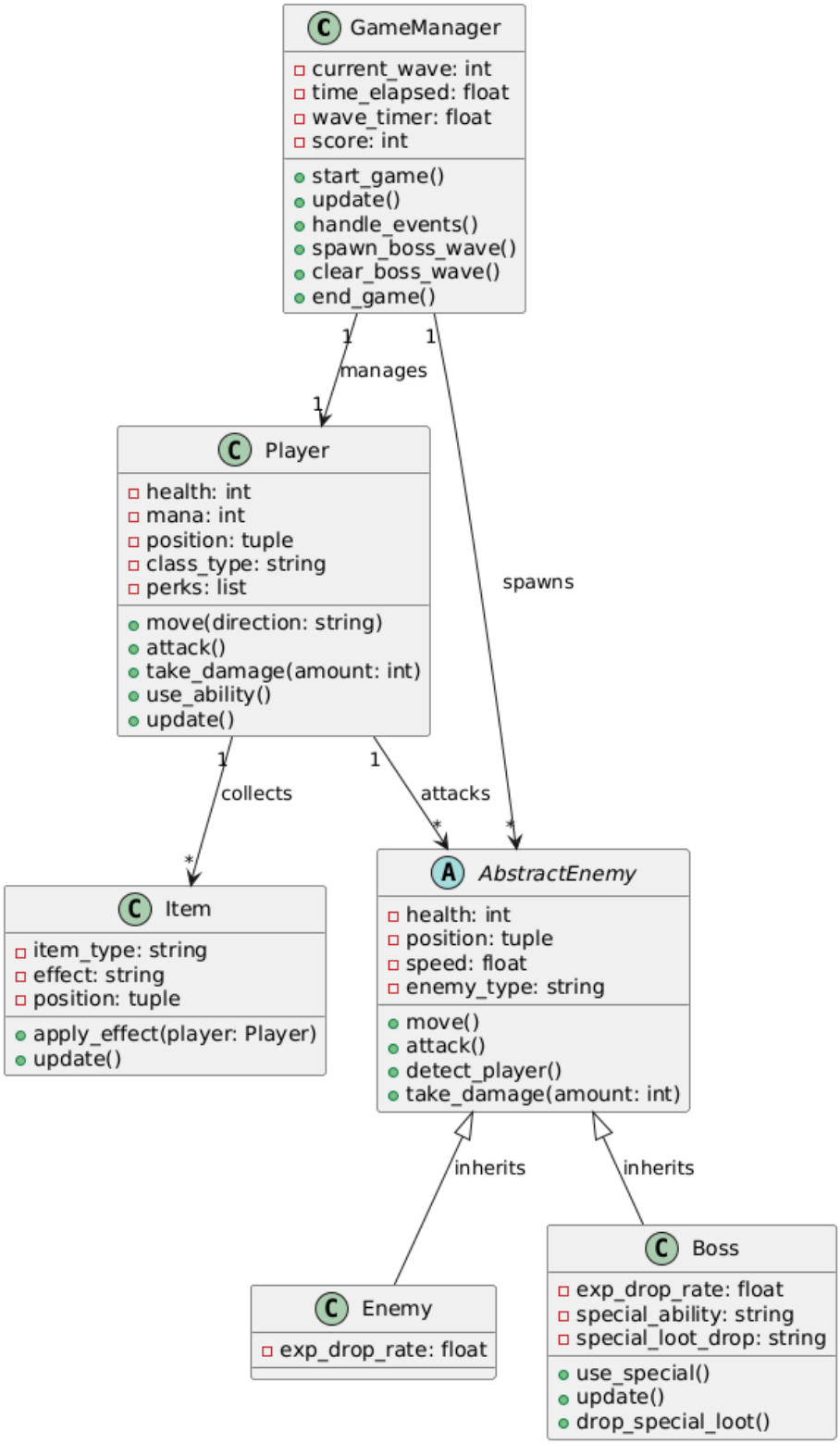
Key Attributes:

- item_type
- effect
- position

Key Methods:

- apply_effect(player)
- update()

UML Class diagram:



3.3 Algorithms Involved

This Game AI behavior will be quite simple since there are an endless number of enemies until time runs out. the enemy will be dealing damage once their character comes into contact with the player.

The Algorithm that will be used for this game is an A* pathfinding algorithm. A* finds the shortest path between two points on a grid, taking into account the cost of movement and any obstacles. It's well-suited for real-time games due to its balance between performance and accuracy.

4. Statistical Data (Prop Stats)

4.1 Data Features

Player Movement Metrics:

Data Points: Player position coordinates over time, distance traveled per minute.

Keystroke Analytics:

Data Points: Frequency and types of keys pressed for actions such as movement, attacking, and ability usage.

Player Score & Progression:

Data Points: Score increments, score loss (if applicable), and score progression correlated with wave advancements.

Enemies & Bosses Defeated:

Data Points: Number of enemies and bosses defeated per wave, including timestamps.

Survival & Wave Completion Times:

Data Points: Time taken to defeat each boss wave, reaction times between waves, and overall survival time metrics.

4.2 Data Recording Method

The game will record statistical data in real time using a lightweight data logging system. The primary methods include:

CSV Files: Log data on each game session to CSV formats, making it easy to export and analyze data.

In-Memory Data Structures: During gameplay, data will be captured in pre-defined data structures (such as dictionaries or Pandas DataFrames) which are periodically flushed to persistent storage.

4.3 Data Analysis Report

Aggregate Data: Use Python libraries (e.g., Pandas) to load and preprocess the recorded gameplay metrics.

Apply Statistical Measures: Calculate means, variances, and correlations between various gameplay factors (e.g., keystroke frequency vs. player survival) to gauge performance metrics.

Visualize Findings:

1. Player Movement Trends:
    - Uses bar plots to represent player movement trends
2. Enemy Defeat Patterns:
    - Represent enemy defeat counts in each wave using bar plots
3. Relationships Between Attributes:
    - Player Movement Trends and Enemy Defeat Patterns: Identify whether increased movement correlates with higher enemy defeat rates
    - Player Movement Trends and Player Survival Durations: Identify whether increased movement correlates with higher duration of survival
    - Number of Player's Perks and Enemy Defeat Patterns: Identify whether number of player's perk correlates with higher enemy defeat rates

Features	**Why is it good to have this data? What can it be used for?**	**How will you obtain 50 values of this data?**	**Which variable (and which class) will you collect this from?**	**How will you display this feature (via summarization statistics or via graph)?**
Player Position Coordinate (Player's Movement trend)	Useful for analyzing movement patterns and strategies. Can identify if players explore or staying still, impacting survival.	Record the player's position every 10 seconds for a 15-minute game (900 seconds / 10 = 90 samples.)	position(from Player class)	Graph: Scatter plot of coordinates to visualize movement trends.
Damage output per perk	Measures the effectiveness of each perk. Can help balance perks by identifying which ones contribute most to damage.	Record damage dealt each time a perk triggers an attack. enemy.	Custom counter in Player class (via attack() method, tracking damage per perk)	Statistics: Mean and variance of damage output per perk type.

Survival time per session	Evaluates overall player performance and game difficulty. Can correlate with other metrics like movement or perks.	Record survival time at the end of each game session	time_elapsed (from GameManager class)	Statistics: Mean and standard deviation of survival times.
Item collection count	Tracks resource management and exploration. Helps assess if items impact survival or combat effectiveness.	Record each item collected; collect data from 50 item pickups across multiple games .	Custom counter in Player class (via apply_effect() method in Item)	Graph: Histogram of item collection counts
Enemies Defeated Per Wave	Helps identify difficulty and player skill growth.	Record enemies defeated in each wave; collect data from 50 waves across multiple game sessions (Maximum of 5 waves per game session)	Custom counter in GameManager class (via update() method)	Graph: Bar plot of enemies defeated per wave to show patterns.

Presentation via Table:

Feature: Damage Output per Perk

Statistical Value: Mean and Variance

- Mean: Represents the average damage dealt per perk across multiple attack instances, useful for assessing perk effectiveness and balancing gameplay.
- Variance: Indicates the consistency of damage output, showing whether a perk's performance is stable.

Feature: Survival Time per Session

Statistical Value: Mean and Standard Deviation

- Mean: Provides the average duration players survive in a session, reflecting overall performance and game difficulty.
- Standard Deviation: Measures the variability in survival times, offering insight into consistency across sessions and potential difficulty spikes.

Graph Category	**Feature Name**	**Graph Objective**	**Graph Type**	**X-Axis**	**Y-Axis**
Relation	Player Position Coordinate (Player's Movement Trend)	Display the player's position coordinates recorded every 10 seconds over a 15-minute session.To visualizes movement trends, such as exploration paths with each point representing a position (X, Y)	Scatter plot	X - Coordinate	Y - Coordinate
Time-series	Enemies Defeated Per Wave	Show the number of enemies defeated in each wave across multiple sessions To highlight difficulty and player performance trends over time.	Bar Graph	Wave Number	Number of Enemies Defeated

Distribution	Item Collection Count	Record survival time at the end of each game session	Histogram	Number of Items Collected	Frequency

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Project Proposal Version 1.1 submission
3 (24 March)	Project Proposal Version 2.0 submission (UML and Data Collection)
4 (31 March)	Full Project Proposal submission
5 (7 April)	
6 (14 April)	Submission week (Draft)

6.Project Planning

26 March - 2 April

- Implement the GameManager class, setting up the game loop, event handling, and wave timing.
- Begin implementing the Player class with basic attributes (health, mana, position) and methods (move, attack).
- Set up the data logging system using CSV files to record gameplay metrics.

3 April - 9 April

- Complete the Player class, integrating four character classes (Mage, Martial Artist, Conjurer, Vampire) with unique perks.
- Implement the AbstractEnemy, Enemy, and Boss classes, establishing inheritance and adding boss-specific abilities.
- Start implementing the A* pathfinding algorithm for enemy movement toward the player.
- Test basic player-enemy interactions to ensure functionality.

10 April - 16 April

- Finalize the A* pathfinding implementation, optimizing enemy navigation on the map.
- Implement the Item class with spawning mechanics and effects (e.g., health potions, power-ups).
- Set up the wave system, spawning bosses every 3 minutes, and enforce the 15-minute survival goal.
- Begin implementing the Awakening mechanic, enhancing class abilities at level 15 (e.g., 20% damage increase).

17 April - 23 April

- Complete the Awakening mechanic, ensuring all classes receive appropriate upgrades.
- Finalize all game mechanics (e.g., combat, item collection, wave progression) and conduct initial playtesting.
- Complete data collection setup for all features: player position (every 10 seconds), damage output, survival time, item count, and enemies defeated.
- Start collecting data by playing the game or using bot to play the game as per TA's suggestion, targeting at least 50 samples per feature.

24 April - 30 April (Start of Exam Weeks)

- Continue data collection, playing additional sessions to meet the 50-sample requirement.
- Begin data analysis.
- Start creating the three graphs (Scatter Plot, Bar Graph, Histogram) using Matplotlib or hand-drawn sketches.

1 May - 7 May (During Exam Weeks)

- Finalize data analysis, ensuring all statistical values are calculated and interpreted.
- Complete graph creation, refining visuals with proper labels and scales.
- Begin testing the game for bugs (e.g., pathfinding errors, perk imbalances) and gameplay balance.

8 May - 11 May

- Fix identified bugs and optimize game performance (e.g., reduce lag in enemy spawning).
- Refine game balance based on data insights (e.g., adjust perk damage if variance is too high).
- Complete the documentation, including the statistical report with tables and graphs.
- Prepare and submit the final project package by 11 May.

7. Document version

Version: 3.1
Date: 31 March 2025

Date	Name	Description of Revision, Feedback, Comments
15/3	Phiranath	Don't forget to format this document again (italic and topic number). I would suggest that for the visualize finding part it would be better if you choose to show information in each attribute using one type then find relation between each attribute for example, use bar plot for Player movement trends then, find relationship between Player movement trends and Enemy defeat patterns. Very detailed documents, good job!
16/3	Rattapoom	Some comments on OOP Implementation.
28/3	Phiranath	I would suggest collecting player accuracy, for example, attacking 40 times and hit only 20 times, so accuracy = 50%. Overall the proposal is good, only some comment on format, and data features.