Computer Games Development
Project Report
Year IV

Ruslan Gavrilov
C00273521
13/05/2024

Project Abstract

This project presents the development of Pirates' Gambit, a 2D grid-based strategy game created using C++ and SFML, centered around tactical turn-based combat, Open world exploration , and resource/army management. The game is designed as a hybrid between traditional grid-based gameplay and real-time responsiveness, requiring players to explore the world, collect money and build up their armies, and strategically engage enemies to survive increasingly difficult fights. Central to the gameplay is a tile-based map system with varying terrain types and node properties, which influence player actions and enemy behaviors. Advanced algorithms, including Cellular Automata for terrain generation and Wave Function Collapse for terrain smoothing, were implemented to procedurally generate diverse island landscapes. The project showcases modular class design, efficient memory management with smart pointers, and object-oriented architecture. Emphasis was placed on both gameplay mechanics and technical execution, ensuring scalability and performance. This report documents the iterative development process, challenges faced, and solutions employed to deliver an engaging and technically sound game prototype.

Project Introduction

The aim of this project was to design and develop a fully functional 2D strategy game titled, which combines procedural world generation, turn-based combat, and dynamic enemy behavior within a well simulated pirate-themed environment. This project was undertaken as part of the final year Computer Games Development module and focuses on the practical application of game development techniques including terrain generation, AI-driven gameplay, and modular architecture.

At the core of the game lies a procedurally generated world composed of interconnected nodes structured in tile-based chunks. These nodes form the foundation of the game's map system, allowing for efficient traversal, rendering, and dynamic interactions. Multiple algorithms were explored and implemented to create realistic and varied island shapes, such as the Cellular Automata algorithm for organic landmass formation. To further enhance realism and immersion, the game features a tile-based texturing system using a customized Wave Function Collapse approach, as well as object placement logic inspired by Poisson Disk Sampling for populating islands with trees, barrels, and buildings.

Gameplay primarily includes exploration, interaction with AI-controlled enemies, and turn-based battles managed by a custom initiative system. Players can recruit allies, build armies, and engage in combat that uses A* pathfinding, dynamic unit behavior, and a responsive UI system. The game also introduces a reputation system that governs how enemies react to the player, contributing to an evolving and strategic gameplay experience.

The development process prioritized performance optimization, and scalability through the use of singleton patterns, updateable areas for pathfinding, and selective rendering. This

report details the design rationale, algorithms used, system architecture, and the various technical challenges overcome to realize a complete and engaging gameplay experience.

Literature Review

I chose to embark on this specific type of game as I have always enjoyed tactical turn based combat games. I took a lot of inspiration from the Heroes of Might and Magic (Link in references) franchise on how I would handle a turn based system to its best efficiency. Understanding those game systems allowed me to create a smart army system as well as an easy to understand combat system that any player of those games would come to immediately recognise.

I also took inspiration from Open World Pirate themed games like Sea of Thieves(Link in references). Sea of thieves allows the players to sail between oceans and land on islands to possibly explore / loot them. I wanted to incorporate this feeling of being a pirate into my game along with the combat and strategic system described above and combine them into an immersive and themed game.

Procedural generation was also an aspect of game development that I had not looked into and with games like minecraft and terraria creating dynamic worlds everytime a new world was created. It inspired me to attempt to create my own custom system of how believable islands would be generated and how these islands would work together inside of my game world.

Project Milestones

This section outlines the major development milestones achieved throughout the creation of Pirates' Gambit. Each milestone marked a significant step forward in gameplay functionality, system integration, or user experience.

Island Generation

The foundation of the game world was established through a procedural island generation system. Early experimentation began with the Diamond-Square algorithm to simulate natural terrain, but the final implementation adopted Cellular Automata to produce more varied and organic island shapes. A tile-based chunk system was introduced to improve performance and structure, enabling efficient updates and terrain management.

The creation and completion of this system allowed me to have the important starting point to build off of and set a pace for the project and for the features to come. It was the first major milestone inside of the project and gave me confidence to continue with developing more advanced features inside of the project.

Battle System

A turn-based battle system inspired by games like *Heroes of Might and Magic V* was implemented for combat interactions. This system includes initiative-based turn ordering, A* pathfinding for unit movement, and action handling for melee and ranged units. A dedicated battle scene was created specifically for the combat mechanics and to manage transitions between player and enemy turns.

Turn based combat was set to be the major strategic point of the project. The combinations of pathfinding, enemy evaluation and the completion of the initiative system allowed my game to be more than a procedurally generated simulation and added depth and purpose to the game.

Enemy AI

A finite state machine controls enemy behaviors such as chasing, idling, wandering, and following the player. Enemies use A* pathfinding and decision-making logic to intelligently pursue objectives. A reputation system was also implemented to dynamically alter enemy interactions based on the player's standing, enabling both hostile engagements and allied cooperation.

Differing enemy states allowed the over world to feel more alive as the enemies could board/disembark their ships, avoid each other, chase the player etc. I believe a key aspect of open world games is to create a believable environment where the entities have a seamless relationship with the player and each other, and the introduction of states and the changing of them along with the reputation system allowed for a more immersive world as well as strategic depth with the ability to recruit pirates / avoid factions, grow your army to defeat more difficult pirates and overall progress through the game.

Island Population

To enrich the game environment, a system for procedurally populating islands with trees, barrels, and buildings was created. Tree placement utilised a disk sampling-inspired method to distribute clusters naturally. Barrels and buildings followed similar logic, with buildings requiring larger clearings. Each object became interactable, contributing to resource collection and unit recruitment systems.

Creating trees only further gave depth to the game and gave a reason for the exploration aspect of it. Rather than bare islands than now just generated, the added foliage and loot and potential of increasing your army gives the player motivation to sail around and discover the map to reach their ultimate potential.

UI / Animations

A responsive UI was developed for various game systems including inventory, battle management, and building interactions. Key interfaces are managed through singleton classes to ensure centralized and efficient control. Animations were added to battle indicators, damage numbers, and turn order transitions, enhancing clarity and player immersion. A loading screen and minimap further contribute to a polished user experience.

Creating fluid movement in the game with animations, UI elements and possible transitions allowed for the game to be better to look at. After the major game systems were developed, they would come across a lot less impressive with static sprites and no life in the game. Adding custom particles like leaves and trails as well as sprite animations and animated UI elements like the initiative bar allows for a satisfying experience to play while the complex systems function in the background.

Major Technical Achievements

Procedural World Generation
Implemented a fully procedural 2D tile-based world made up of multiple chunks, each with 32×32 nodes. Nodes were structured with interlinked neighbors to support efficient pathfinding and traversal. Chunking allowed optimizations by reducing the search space for AI and rendering systems.

Island Generation Using Cellular Automata
Evaluated and implemented multiple terrain generation algorithms including the Diamond-Square and Cellular Automata algorithms. Ultimately selected Cellular Automata for its ability to produce more natural, non-circular, and varied island formations, supporting multiple islands per chunk.

Context-Aware Tile Texturing via Wave Function Collapse
Designed a custom tile-based adaptation of the Wave Function Collapse algorithm to assign textures dynamically based on land/water neighbor nodes. This allowed for seamless transitions between terrain types.

Dynamic Island Population Using Disk Sampling Principles
Populated islands with environmental objects (trees, barrels, and buildings) using a custom distribution algorithm inspired by Poisson Disk Sampling. Ensured spatial coherence by using breadth-first searches and clump formation with occupancy checks.

Turn-Based Battle System with Initiative Management
Developed a tactical battle system modeled after *Heroes of Might and Magic*, featuring unit stats, pathfinding (A*), and turn-based combat. The system includes an initiative queue that dynamically adjusts unit turn order based on actions taken and remaining initiative points.

AI Decision-Making and Enemy Behavior States
Created a class-based state machine for enemy units with behaviors including Idle, Chase, Wander, FindBoat, and FollowPlayer. Integrated pathfinding and obstacle avoidance into AI movement and decision-making logic.

Dynamic Enemy Loading via JSON
Enabled runtime enemy instantiation through structured JSON files. Each enemy is loaded with individual stats, names, and armies, allowing for scalable and easily adjustable enemy deployment.

Selective Rendering and Update Optimization
Implemented a visibility-based rendering system that calculates visible nodes using the camera's position. Only visible nodes and nearby game objects are updated/rendered, significantly improving performance on large maps.

Minimap with Progressive Discovery
Built a dynamic minimap that reveals terrain as the player explores. Initially blacked out, tiles are revealed and color-coded based on their type (land, water, sand), enhancing exploration feedback.

Expandable Inventory and Interaction Systems
Designed an inheritance-based inventory system that supports stackable items, transfer between entities (e.g., barrels to players), and in-world interactable structures like unit-producing buildings.

Expandable Army System
Designed an inheritance-based army system that supports stackable units, transfer between armies, and custom Armies for each pirate unit loaded via JSON.

Custom Animations
Designed custom animations for game immersion such as the initiative bar self animating when updating to a new turn.

Singleton-Based UI and Game Systems
Utilized singleton patterns for core systems (e.g., Camera, Mouse, Animator, UI managers), streamlining access and reducing redundancy in rendering and interaction logic.

Particle and Bullet Management
Developed centralized factory and manager classes for handling game objects such as particles (e.g., effects) and projectiles (e.g., cannonballs), ensuring clean updates and memory management.

Project Review

What Went Right

The procedural world generation was a major success in this project. Breaking the world into chunks and generating individual islands within those chunks provided a scalable and efficient approach to terrain generation. Cellular Automata proved to be the most effective algorithm for island shaping, producing more varied and natural results compared to Diamond-Square. The use of Wave Function Collapse for terrain texturing gave the islands a visually coherent and polished appearance.

The turn-based battle system was another strong point. Integrating pathfinding with initiative-based unit management created dynamic and strategic combat similar to classic tactics games. The AI's state machine design allowed for predictable behavior, supporting future extensibility. Additionally, loading enemies and their armies from JSON files proved invaluable for scalability and testing.

Optimizations such as selective rendering and the updateable area system significantly improved runtime performance and made the large-scale world practical. The use of singleton patterns for commonly accessed systems (UI, Camera, Mouse) also streamlined the codebase.

What Went Wrong

Several challenges arose with texture assignment and terrain validation during procedural generation. Some tile configurations led to broken or awkward terrain visuals that required post-processing logic to identify and correct invalid patterns. Additionally, the Diamond-Square algorithm was initially pursued for terrain generation but had to be abandoned due to producing overly uniform, centralized island shapes, highlighting some lost development time during experimentation.

In the battle system, issues occurred with turn order logic, especially when multiple units from the same army acted in succession. Managing unit states and handling transitions (particularly deaths) without breaking the initiative flow required additional layers of logic and debugging.

Enemy AI pathfinding had to be throttled to avoid performance issues, especially when multiple agents tried to compute paths at once. Furthermore, placing large buildings on densely populated islands proved unreliable and required extensive trial-and-error logic with hard-coded attempt limits.

Outstanding/Missing Work

While the procedural and combat systems are functional, several areas remain for future development:

- **Combat variety and depth**, such as abilities or terrain-based effects, could enhance strategic options.

- **More robust placement** logic could reduce the number of failed placement attempts and increase consistency.

- **UI polish and in-game tutorials** are still needed to guide new players through the mechanics.

What Would Be Done Differently

If starting again, more time would be allocated to researching and prototyping terrain generation algorithms before committing to implementation. Cellular Automata turned out to be the best fit, but the time spent on Diamond-Square could have been avoided with more up-front validation through smaller tests.

The unit and AI systems might benefit from using a dedicated behavior tree or utility AI system instead of a hand-rolled state machine for better scalability and maintenance. Similarly, the UI system might be abstracted earlier using an external library or framework to speed up iteration and reduce the reliance on manual layout and interaction logic.

Advice for Future Projects

For anyone attempting a similar project:

- Start with **modular systems**; break everything into reusable and testable components (e.g., pathfinding, generation, AI).

- Avoid hard-coding values where possible—use external data (like JSON) for scalability and iteration speed.

- Implement **debug visualization** tools early. Seeing your map, AI state, and node relationships helps catch logic errors.

Technology Choices

The technology choices for this project were largely appropriate:

- **C++ with SFML** provided enough flexibility and control for the low-level systems like pathfinding and procedural generation. The use of pointers and memory control allowed for optimization, and custom low level technical achievement.

- **JSON file handling** for enemy definitions and armies made it simple to scale and test different configurations.
  .

However, if the project were to be scaled up, switching to an engine like Unity or Godot may be more appropriate. These engines offer built-in solutions for UI, pathfinding, animation, and asset management that would accelerate development. SFML, while lightweight and flexible, lacks high-level features and requires manual handling of many game-engine responsibilities.

Implications of Technology Choices

Using C++ and SFML gave full control over memory and performance, which was critical for implementing efficient procedural systems and low-level optimizations like selective rendering. However, the cost was increased development time and complexity, every feature, from pathfinding to UI interaction, had to be built from scratch.

In summary, the project demonstrated a solid grasp of systems design and implementation, with procedural generation, combat logic, and AI systems working in unison. While some technical and design challenges were encountered, they were successfully resolved, and the final result is a promising prototype with room for expansion.

Conclusions

Pirates' Gambit represents the outcome of my final year project that integrates procedural generation, strategic gameplay, and intelligent enemy behaviors into a structured, seamless and engaging game experience. The project demonstrates the effective application of object-oriented programming, algorithm design, and performance optimization using C++ and SFML.

Through experimentation with world generation techniques, such as the Diamond-Square and Cellular Automata algorithms, an intelligent and varied terrain system was achieved. Coupled with procedural island population and chunk structure, this laid the foundation for a dynamic game world. The integration of a turn-based battle system, complete with initiative management and enemy AI, added strategic depth, while supporting systems like inventory management, reputation tracking, and UI components helped round out the gameplay loop.

Challenges encountered during development, particularly in synchronizing systems like AI decision-making and world generation, provided valuable learning experiences in debugging, system coordination, and feature scalability. Ultimately, the project not only met its functional goals but also highlighted areas for future work and refinement.

This project has deepened my understanding of complex game systems and reinforced my capability to manage a large-scale software development cycle from concept to execution. It serves as both a technical achievement and a creative expression of my skills as a developer.

Future Work

There are many areas that can be expanded or refined to enhance the overall gameplay experience and technical depth. Future development could focus on the following key aspects:

1. **Quest and Story System**
   I would add in a dynamic story into the game that gives the player further purpose than beating all of the pirates. Giving each enemy a sense of character, unique dialogue options and a more dynamic system of choosing sides and deciding morally how you want to play the game would increase immersion and add more character and life to the game.

2. **Enhanced AI Behavior**
   Current AI uses a state machine to control enemy behavior, but further development could include learning-based systems, such as decision trees or behavior trees, to make enemy responses more dynamic and context-aware. Enemies may interact with each other, form fleets do make turn-based combat even more challenging or interact with buildings to stop the player from recruiting possible extra units.

3. **Expanded World Content**
   Additional procedural content such as weather effects, and dynamic world events (e.g., raids or storms) would enrich the game world and provide unpredictable challenges. The ocean aspect of the game is currently quite large and populated by ships. Given extra time, I would incorporate possible quicktime ocean event like shark

attacks, floating treasure for extra loot etc.

4. **Expanded Army/Inventory**
   With the modular system that is currently in place for army units and possible inventory items, it would be easy to add in new possible items/ army units to the game to give more gameplay options or more dynamic battles. New pirate units could be added to give the turn based system more strategic depth and more items like new cannon balls, potential artifacts that increase combat could be added as well to make exploration more rewarding and the gameplay more diverse.

5. **Audio and Visual Polish**
   Adding background music, ambient sounds, and improved animations would significantly boost immersion.

References

Web-site

Craft of Coding. (2021, July 12). The Square-Diamond Algorithm for 2D Surfaces I: Basics. [Blog post]. Retrieved from https://craftofcoding.wordpress.com/2021/07/12/the-square-diamond-algorithm-for-2d-surfaces-i-basics/

Sebastian Lague. (2019). Coding Adventure: Wave Function Collapse [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=slTEz6555Ts

The Coding Train. (2018). 10.6: Diamond-Square Algorithm - Nature of Code [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=dFYMOzoSDNE

sighack. (n.d.). Poisson Disk Sampling: Bridson's Algorithm. Retrieved from https://sighack.com/post/poisson-disk-sampling-bridsons-algorithm

Heaton, R. (2018, December 17). WaveFunction Collapse Algorithm. Retrieved from https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/

Sea of Thieves Wiki. (n.d.). *Sea of Thieves Wiki*. Retrieved from https://seaofthieves.wiki.gg

Heroes of Might and Magic Wiki. (n.d.). *Heroes of Might and Magic V*. Retrieved from https://mightandmagic.fandom.com/wiki/Heroes_of_Might_and_Magic_V