Knight's Quest: The Goblin Heist !
Group 16

Group members:
Manya, Will, Tom, and Pardeep

Origin

- NES style game format.
- Inspiration from Classics such as Legend of Zelda.

Legend of Zelda: 1987

Knight's Quest: The Goblin Heist

How to Play?

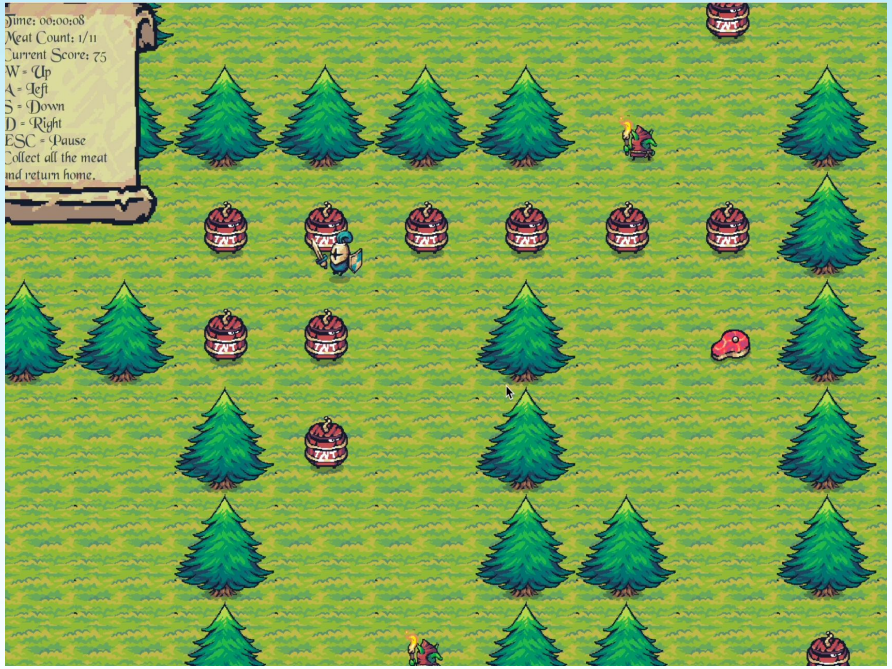
- Goal is to collect 11 meat objects and get back to home base successfully.
- Ran into an enemy goblin; you instantly die and have to restart game.
- Ran into TNT; you lose some of your score, but if you have a score of 0; you die on impact.

- There are optional gold pouches around the map to increase your score, helping you to survive contact with the enemy.

Game Video

Fun facts about Game

- Responsive AI, in terms of goblin, it follows you around.
- Implemented animations for Knight, TNT, and Goblin.
- Camera tracking which follows knight as he traverses the map.



Management of the Project

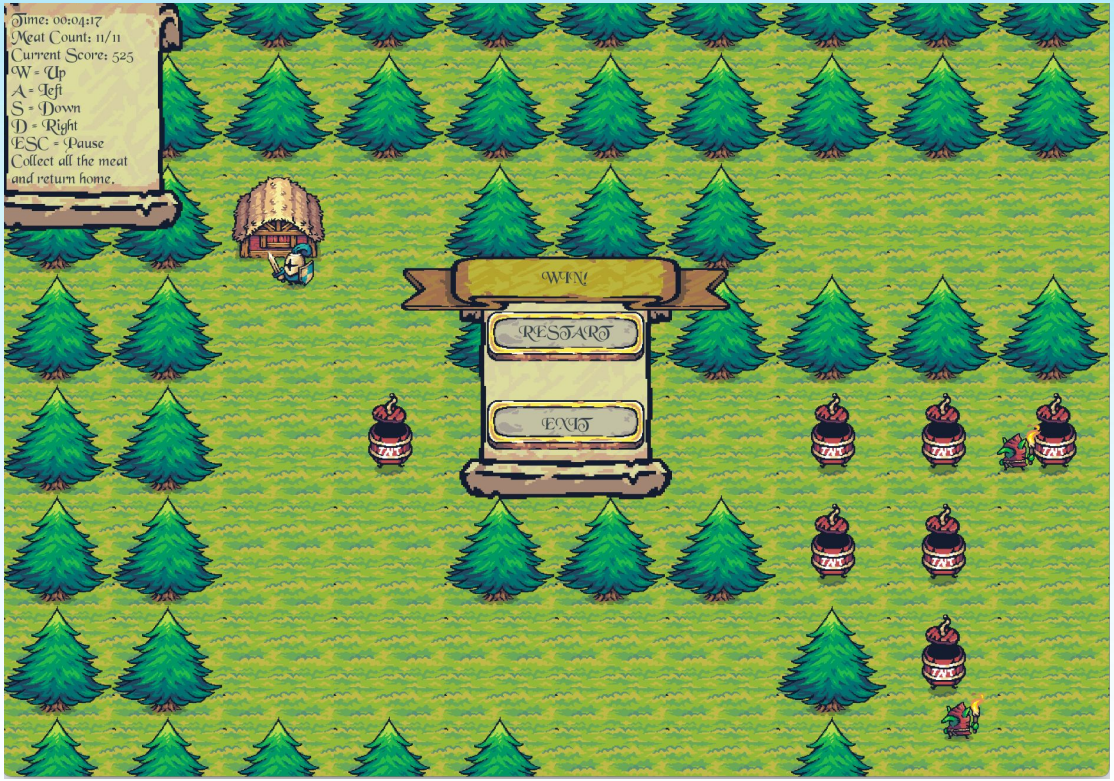
Scrum!

By adopting Scrum, we aimed to streamline project management processes, enhance team communication, and deliver valuable increments of work at regular intervals.

With its emphasis on frequent feedback loops and continuous improvement, Scrum provided the framework necessary to navigate uncertainties, respond to changing requirements, and ultimately ensure the successful delivery of your project objectives within the desired timeframe.

Game Design

- Entities
- States
- Settings
- Control
- UI



Game Design: Entities (representing objects)

Entities: Tree, Coin, Dynamite, Knight, Goblin, ...

- Listed in terms of increasing complexity. Simple ones just render to the screen.
- Complicated ones add animations, movement, and player interaction.
- Invisible Hitboxes (rectangles) represent actual Entity positions in the world.
- Attributes: Height, Width, Images, ...
- Methods: Rendering, Animation updates,...

Interfaces: Mover and Interactable

- Mover facilitates movement for complex Entities.
- Interactable allows for Entity interaction through hitboxes.

Game Design: Game States

Enum: Menu, Playing, Pause, Win, Defeat: Different states game can be in.

Game Class:

- Switch between game states. Render and update current state.
- Draw according UI to the Screen
- Connects to keyboard and mouse inputs

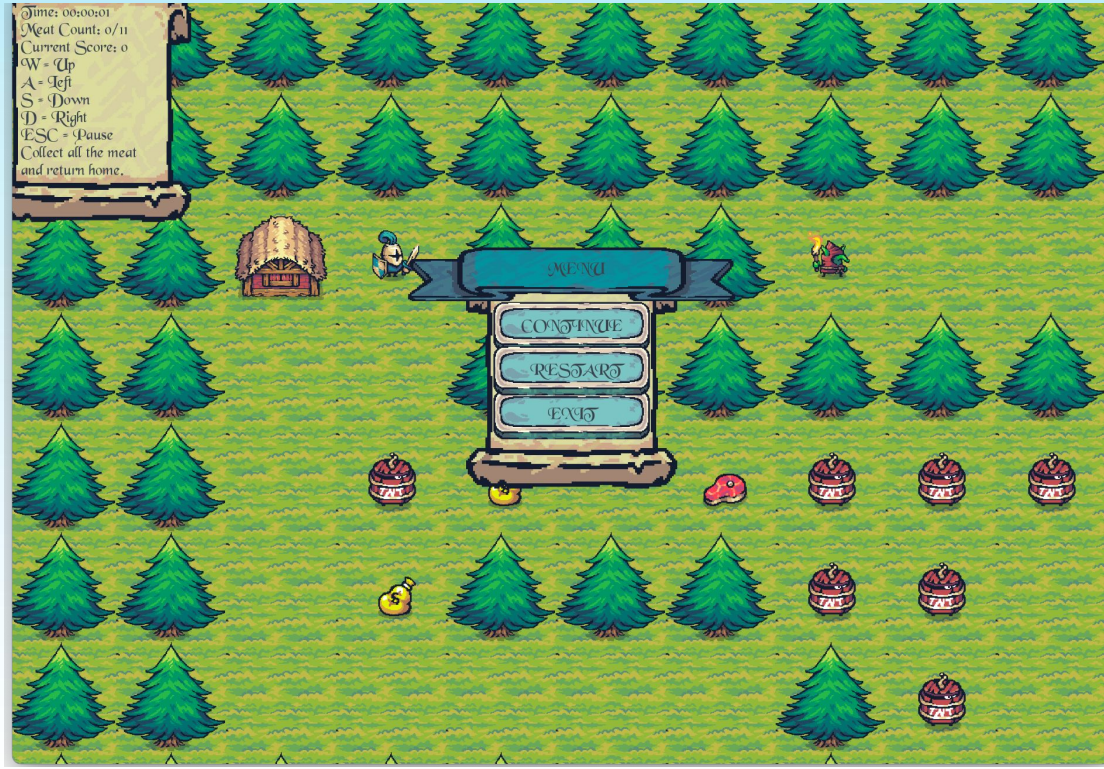
Game Design: Settings

Screen: Show the game world.

Camera: Show a portion of the world relative to the knight. Other objects are offset to it.

Timer: Show time in game. (up left corner)

Scoreboard: Show updating score



Game Design: Control

- **Keyboard Control:** Player movement.
- **Mouse Control:** UI interaction.

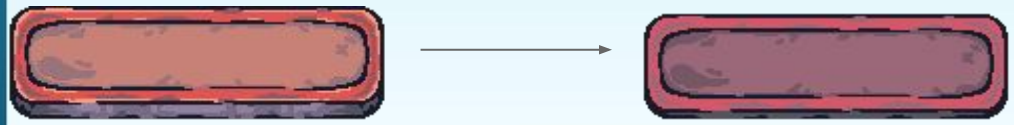
```
import java.awt.event.MouseEvent;

import java.awt.event.KeyEvent;
```

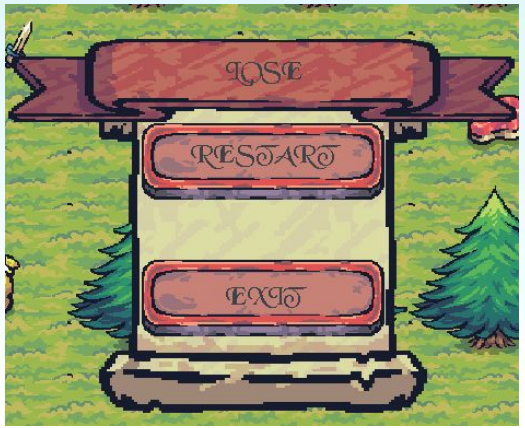
- **AI Movement:** Implemented A* search algorithm for the Goblin to track the knight. Only triggers within a certain range of the Goblin.

Game Design: UI

UI Buttons

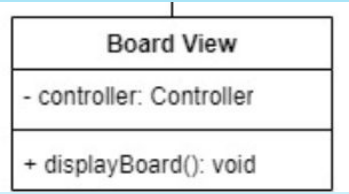
- Design the appearance of each button.
- Have interactable buttons (feedback of press and release).
- Able to trigger changing state according to the certain button.

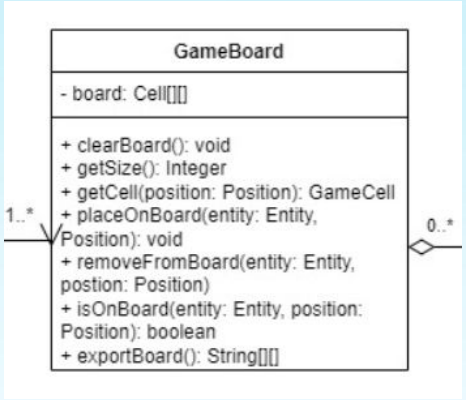




Problems

- Portion of our Phase 1 UML Diagram.
- Group Java experience initially: close to zero.
- Did not know how rendering worked, how it interacted with different game components such as the internal game world.
- Abstracted it away in Board View class.
- Chose Integer based grid world made of cells for internal game world.





Problems

- Asset pack we chose for our project, highly suited for floating point world.
- Not suited for our integer based world.
- Complicated enemy movement and world creation.
- Reconciling the two caused a large delay in our first milestone, which was rendering/world creation based.

TinySwords by Pixel Frog on itch.io
https://pixelfrog-assets.itch.io/tiny-swords

Problems

- Scrum is an amazing framework for managing a group's output and milestones.
- However, it needs to be implemented and enforced correctly to be useful, the same as any other project management framework.
- Due to scheduling conflicts that occurred prior to the halfway deadline, serious work only began relatively close to the deadline.Combined with the delay previously mentioned, features had to be cut. Attacking, killable enemies, respawn points for enemies, special power ups, etc.
- For future milestones and phases, principles of scrum, enforced meeting times, and strict output requirements were enforced to put group production back on track.

Refactoring

- During Phase 2, large amounts of unrelated tasks were assigned to Entities (animation, pathfinding, collision detection, large amount of unrelated static variables for random information). This decreased cohesion, increased coupling.

- Similarly, State classes were also highly coupled together. Changes in one necessitated changes in multiple others.

- How it was fixed: Removed all rendering capabilities from the Entities. Placed them into their own classes. Entities only drew a static image of themselves. Pathfinding for enemies also placed into its own class.

- State classes were made to communicate through the Game class, which manages and switches the current state depending on input.

Tests

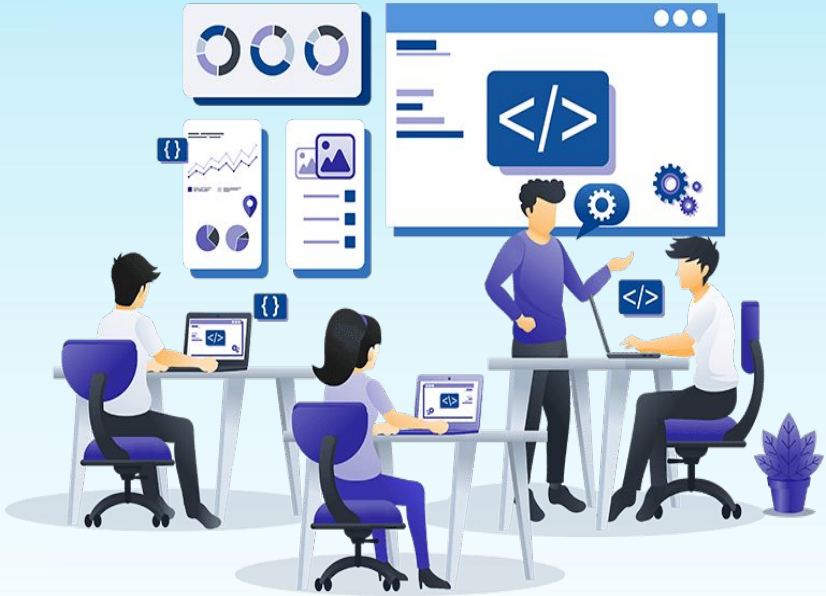
Line Coverage 90%

Branch Coverage 71%

- Branch coverage was low.
- Certain branches difficult to replicate through unit or integration testing alone.
- Further refactoring required to increase coverage.

Stuff We Learned

- Git
- Java
- Dependency Management
- Code Organization
- Software Design Principles
- Debugging
- Testing
- Continuous Integration
- Code review
- Documentation
- Time management :(



Lessons Learned: The Hard Way

- When you get told that things get expensive to change and fix later on, believe it.
- Have a workflow that is adaptable to change. Avoid the waterfall.
- Constantly communicate, especially when problems appear. Don’t disappear.

Why did the software development team go to the team-building workshop?

Because they heard it was a great place to "debug" their teamwork!