WaS-Overview

Elijah T. Rose

November 13, 2019

TO	Gregory Myers (gmyers@uab.edu)
CC	Reece Younger (ryounger), Alex Norman
FROM	Elijah T. Rose (elirose)
START	2019-11-01
END	2019-11-13
PROJ	Wolves and Sheep
SUBJ	Overview-V2

EE433 Final Project

In an attempt to further our education and expand into non-trivial software projects, we are developing a multiplayer command-economy and military low-fantasy Real-Time Strategy[1]-esque videogame.

Description

The project will be a real-time strategy[2] game that features a platform agnostic software for portability alongside a designed Windows 3D Renderer view. The game is set in a low-fantasy medieval realm with different feature maps to have different

[1]"Real-Time" is used in reference to strategy games where each player acts asynchronously, as opposed to "turn-based" strategy where each player waits for an active member to act before being able to act themselves. "Real Time" does NOT imply it is truly real-time or instantaneous – indeed, the server and client are necessarily always out of sync, even if all clients and the server are on the same process there will be a small propogation delay, and this is why there are N+1 models or instances of the game running simultaneously even in a N-player game (not accounting for spectators).

Real-Time Strategy, though, is a well-known genre and descriptive term that is near universally acknowledged and it is doubtful that there will ever be a misunderstanding of its term – casual observers and gamers will only recognize the term from its genre connotation, and programmers and software leads, if somehow unaware of that connotation, can likely deduce that it does not imply true real-time as that is effectively impossible by know physics with any sort of propogation delay between instances of a software. It is also for this reason that the more appropriate label Real-Time Tactics is not necessarily applied. Though "Strategy" implies a broad arsenal of interfactional exchanges whereas our game only allows tactical military play, the understanding and connotations of RTS are understood and help convey the message quickly and in terms most intended audience members understand or can quickly grasp, which is more desirable in a project overview over technical and precise terms that may hinder the understanding by most users and even game developers. Technical lanaguage and description, such as calling the game an "Asynchronous Multi-Instance Theoretical-Problem Intercommunicable Partially Zero-Sum Game Simulation Software" has its time and place, but is foolish in trying to give your "sales pitch" with. https://en.wikipedia.org/wiki/Real-time_strategy https://en.wikipedia.org/wiki/Real-time_tactics

[2]"Real-Time" is used in reference to strategy games where each player acts asynchronously, as opposed to "turn-based" strategy where each player waits for an active member to act before being able to act themselves. "Real Time" does NOT imply it is truly real-time or instantaneous – indeed, the server and client are necessarily always out of sync, even if all clients and the server are on the same process there will be a small

gameplay mechanics. Our focus will be a competitive arena like gameplay to see which player can win utilizing a combination of both economy and military. Each player may choose different gameplay routes via a different strategy or reacting to an enemy player's attack or expansion. Unlike most real time strategy games, this game will feature each unit being trainable to multiple tasks within a village. A villager can either be trained for mining, woodcutting, farming, a soldier, defender, or other such task – each unit has their own skill tree and can take on multiple roles over the course of a play session.

Like individual units, players can use resources to build different buildings to improve the economy or train soldiers. Each building will also have its own health associated so that attacking players can destroy key buildings to stunt growth. A player can win a game by destroying all of the enemy players buildings or completeing some other objective. The game will have a client-server model where players can theoretically play worldwide. Each player will have their own account and have a rank associated. The ranking system will make sure that new players are not queued with veteran players given options. Veteran players will have a challenge to reach the top of the ranking system making this game have potential replay value.

Purpose

The purpose of this project is twofold:

Firstly, there is the creation, managememnt, and execution of the game project itself. This largely benefits we, the developers, in being able to demonstrate and practise software engineering principles. It additionally adds to our work-experience and portfolios while potentially being demonstrable to the school at large, that is, a reflection of the EE department. This is the "meta-purpose" or framing outside of the software itself, emergent from its development. Furthermore, while this specific project does not have such features, it is similar to and may enable we developers or lead to other, more practical and educational tools.

Secondly, the game itself seeks to entertain and challenge players. Within the context of the game, players are expected to think within the bounds of given tools and develop approaches, tactics, and pathways to overcome their opponent. To this extent, the game serves almost as an asynchronous, interactive competitive puzzle, allowing player authorship in discovering and solving problems they may face.

Major Features

Gameplay Features

- Control and manage multiple units to do various tasks.
- Gather resources from the world around you.
- Process materials into other materials.
- Craft various tools and weapons to enhance your units.
- Construct buildings with which units can interact.
- Food to upkeep units, and implicitly the large space needed to make food, as pseudo-limit to unit growth.

Technical Features

- Connect to one or more other players based on usernames.
  - Re-connect in lost sessions should both player attempt to reset; periodic save states.
  - Potentially through your Steam or similar networking interface, ie meta friend-list.
- Save states that can be passed and loaded by another copy of the game.
- Updator that allows easy updating to the next version without reinstalling.
- Menu and GUI to access and navigate customization options of the game.
  - Options Menu
  - Find Game
  - Exit Game
  - Load Game

propogation delay, and this is why there are N+1 models or instances of the game running simultaneously even in a N-player game (not accounting for spectators).

Real-Time Strategy, though, is a well-known genre and descriptive term that is near universally acknowledged and it is doubtful that there will ever be a misunderstanding of its term – casual observers and gamers will only recognize the term from its genre connotation, and programmers and software leads, if somehow unaware of that connotation, can likely deduce that it does not imply true real-time as that is effectively impossible by know physics with any sort of propogation delay between instances of a software. It is also for this reason that the more appropriate label Real-Time Tactics is not necessarily applied. Though "Strategy" implies a broad arsenal of interfactional exchanges whereas our game only allows tactical military play, the understanding and connotations of RTS are understood and help convey the message quickly and in terms most intended audience members understand or can quickly grasp, which is more desirable in a project overview over technical and precise terms that may hinder the understanding by most users and even game developers. Technical lanaguage and description, such as calling the game an "Asynchronous Multi-Instance Theoretical-Problem Intercommunicable Partially Zero-Sum Game Simulation Software" has its time and place, but is foolish in trying to give your "sales pitch" with. https://en.wikipedia.org/wiki/Real-time_strategy https://en.wikipedia.org/wiki/Real-time_tactics

    - Profile
- Profile creation and management.
- System-agnostic Controller and Model (distinguished View), implying relatively trivial porting to different views.
- Client-Server Connection Model
    - When lacking a server, the player with the stronger specs (as judged by network connectivity and multi-processing capability) hosts both a client and server – the server is automatically managed and relatively inaccessible by the user except in turning off and on unless a dedicated customizable playsession is activated.
    - One computer can potentially host a server as well as multiple clients, assumming the necessary resources and input devices.
    - Multiple clients can connect to a single server, thus expanding player-count is relatively trivial.
    - Multiple seperate game instances can be run on one server, connecting various clients without overlap of data.
    - Latency-guarding mechanics to ease higher latency delays and connection loss (rubber-banding) – singular "master" model in server, dictates and synchronizes clients.
    - Use of JSON/Serialization to pass Event data between server and clients via Controller.
        - Events as an interface or parent class from which all events inherit; events are serializable, thus the client and server adapters simply encode, pass, and decode controller action that changes Model state.

Stakeholders

- EE433 Instructor (Myers)
    - List of software requirements
    - Seeking to test knowleedge of non-trivial software design and interaction of several components.
    - Desires a well-planned project and project review and presentation (via at least a show-board).
- Electrical Engineering Department
    - Similar to Instructor, invests in the Student-Developers to potentially gain presentable results, showing others what the department teaches and potentially gain new investors/students.
    - Perhaps extendable to UAB as a whole.
- EE433 Students/Developers (Reece, Alex, Elijah)
    - Seeking to learn game designing concepts.
    - Learn advanced software interfaces and facets.
    - Create a well-organized and enjoyable project and product.
- Gamers-RTS Fans
    - Desire multiple gameplay options and choices
    - Player authorship and expression through playstyle
    - Game knowledge and interaction dependent skill.
- Game Publishers/Owners
    - In this case, one and the same as student-developers, but potentiall separate.
    - Manages the hardware and direction of the game.

Languages

- C# - Major game component interactions, object-oriented.
- C++ (maybe) - Steamworks API programming
- C (maybe) - Low-level processes that must be done fast and efficiently

Frameworks

- Unity 3D
- Steamworks Connection API
- SteamworksServer
- Facepunch.Steamworks - helps integrate Steamworks used C/C++) with Unity/C#
- NLog - Logging System for .NET
- DB-OO connection Module (maybe)
- .NET Core
- Windows Development Kit
