AR3

FYP Proposal for

A Turn Based Strategy Game Focusing on Management of Army Logistics

by

HUI Nathan, LEUNG Ho Man Max, LO Yuk Fai, and KONSTANTINO Hubert Aditya

AR3

Advised by

Dr. Sunil ARYA

Submitted in partial fulfillment

of the requirements for COMP 4981/CPEG 4901

in the

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

2021-2022

Date of submission: February 14, 2021

Table of Contents

Introduction 4

- Overview 4
- Objectives 5
- Literature Survey 6

Methodology 10

- Design 10
- Implementation 13
- Testing 14
- Evaluation 14

Project Planning 15

- GANTT Chart 15

Hardware and Software Requirements 18

- Hardware 18
- Software 18

References 19

Appendix A: Meeting Minutes 20

- Minutes of the 1st Meeting 20
- Minutes of the 2nd Meeting 22
- Minutes of the 3rd Meeting 23

1. Introduction

1.1. Overview

The gaming industry has experienced colossal growth over the course of the pandemic. COVID-19 has prompted social distancing measures globally, causing many to find alternative ways to interact with their friends, such as through online gaming. The tremendous growth of the gaming industry can be attributed to the surge in need for online socialization, as well as extra time being spent at home with computer access during lockdown.

Currently, the net worth of the gaming industry is valued at around 300 billion US dollars, surpassing the combined worth of two other entertainment titans, movies and music [1]. The most popular gaming titles pay out millions of dollars to their best players while massive audiences spectate. Currently, first person shooter (FPS) and multiplayer online battle arena (MOBA) games dominate the esports scene. The top eight esports by prize pool in 2020 were all FPS and MOBA games [2].

A notable genre missing from the top is strategy games. Real-time strategy games such as Starcraft were extremely popular at the dawn of esports. However, the popularity and development of strategy games have decreased dramatically. In recent years, most strategy game releases have only been sequels based on previous titles.

While strategy games are not the most popular game genre anymore, there remains a large, dedicated following for them. Most strategy games revolve around creating an army to defeat opponents, whether by strategic positioning or brute force. However, these army units in such games never run out of ammunition or food. Napoleon said, “An army marches on its stomach.” Although it is understandable that strategy games are designed without as much focus on the logistics of managing an army to reduce complexity, it can make a game feel unrealistic.

In this project, our team endeavours to create a strategy game with an equal emphasis on combat and positioning, as well as management of army logistics. We feel that this adds a layer of complexity that is not as prominent in other strategy games.

1.2. Objectives

The objective of this project is to create a strategy game with an equal emphasis on combat and positioning, as well as management of army logistics. The project hopes to achieve these goals by:

- **Creating a logistics management system in gameplay**

  The main focus of our project is to incorporate a system that manages a player's army logistics. Thus, the project will create and implement a logistics management system for a player's army that is not overwhelmingly complicated, but adds strategic depth.

- **Creating an attractive strategy game**

  The game should be interesting and fun to play. Ultimately, the purpose of a game is to derive enjoyment and pleasure. An enjoyable new player experience is paramount, so the learning curve should be gradual. Through the design of different systems, the gameplay should reward players with higher strategic skill.

- **Developing multiplayer features**

  Alongside a singleplayer historic mode, our project hopes to include multiplayer functionality. Pitting player against player generates more fun gameplay.

- **Implementing the above objectives into a smooth game**

  The game should run smoothly, be appealing visually, and have fun gameplay, in order to create an enjoyable user experience.

1.3. Literature Survey

As there are already plenty of strategy games on the market, this Literature Survey will explore and examine the games that most influenced our design and provided inspiration for this project.

Advanced Daisenryaku

Developed by Sega in 1991, Advanced Daisenryaku is a turn-based strategy game that features a series of World War II battles on the European Front. Players play as main combatants in Europe such as Nazi Germany, Great Britain and the Soviet Union.

In the game, players have to control all units they have. It is tedious to micro-manage every unit a player controls, especially during the late game. In light of this, an “auto-pilot” system was adopted for this project to overcome the problem.

Advanced Daisenryaku

Hearts of Iron IV

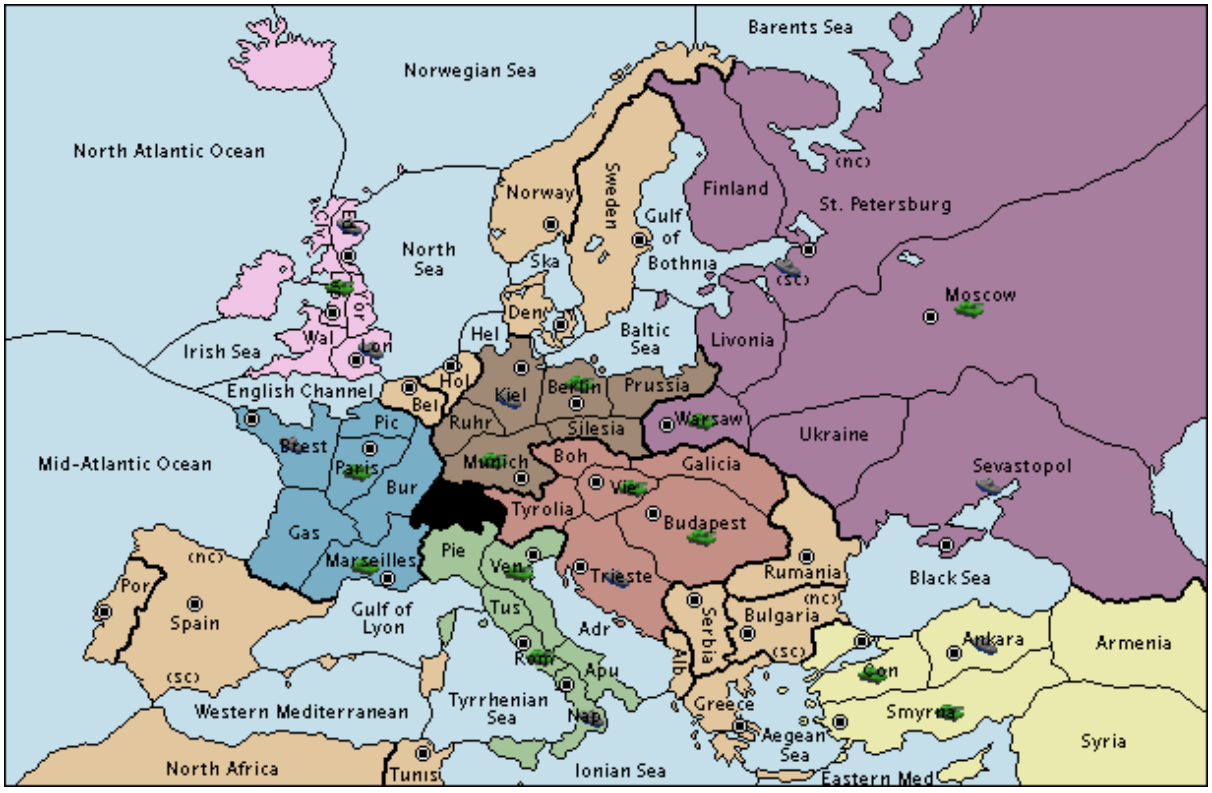
Released in 2016, Hearts of Iron IV is a real-time grand strategy game developed by Paradox Interactive. Players can play as any nation in World War II. The game focuses on various aspects of the War, including military, diplomacy, economy and espionage.

Hearts of Iron IV

Diplomacy

Diplomacy is a strategy board game developed in 1954 by Allan B. Calhamer. Each player represents one of the seven powers in the times prior to World War I. The game focuses on diplomatic relationships instead of military aspects. Players can win the game by negotiating with other players, or even betraying them.

All players take their turns at the same time instead of following a sequence. This is also known as “simultaneous turns”, which can greatly reduce waiting time of all players in each turn.



Diplomacy

Starcraft 2

Starcraft 2 is a fast-paced real-time strategy game developed by Blizzard Entertainment. Unlike other games mentioned in this Literature Survey, Starcraft 2 is played in real time. Real-time strategy games are known more for high mechanical skill and intensity, and less on the actual strategies and tactics.

Since the high mechanical intensity of real-time strategy games can take away from the strategic aspect of the game, a turn-based gameplay format was chosen for the project.

Starcraft 2

2. Methodology

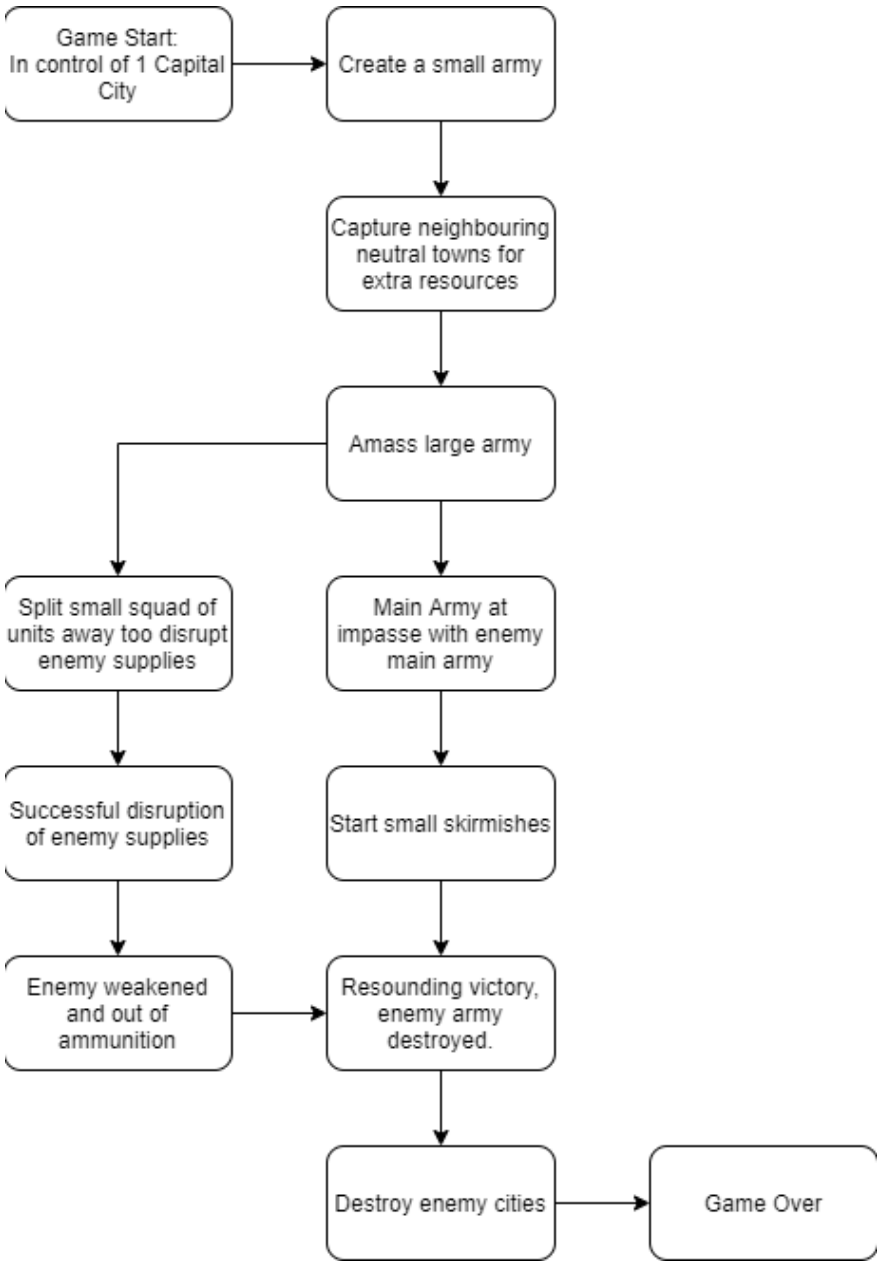
2.1. Design

Our project consists of 4 main parts: Gameplay, User Interface, Network and Bot Algorithm

Gameplay

Each game will have the same objective: to destroy or capture the enemy's cities. The game will take place on a hexagonal game board.

A sample flowchart for the gameflow of a typical game is shown below.



Our game will also include systems to manage resources, logistics, combat, etc. Some of these systems are currently under final stage development and testing.

Resource Systems

Resources will be automatically generated every turn, based on the amount of cities and villages a player owns.

Logistics Systems

Players can flip between 2 viewing modes, logistics mode and combat mode. In

logistics mode, the player can see and manage supply routes, resource collection, and the construction of infrastructure such as unit training facilities. Supplies are consumed by units each turn even if they are holding their position. The further they move, the more supplies they consume. When a unit runs out of supplies, it cannot move until it is re-supplied. The terrains of tiles that the unit traverses also affect its supplies consumption (e.g. crossing a desert terrain will increase supply consumption by 25%).

Unit Creation Systems

There are unit training facilities in cities that will allow players to train different units, depending on which training facilities were built. Training a unit will consume resources and take time, both increasing based on the strength of the unit.

Combat Systems

Each turn, players can designate units to either move, shoot, or ambush. A unit cannot execute more than one of these actions per turn.

A unit ordered to move will move to a designated location, limited by its speed.

A unit ordered to shoot will shoot a target it can see. Shooting at a target moving will halve the damage dealt. Shooting at a target moving out of a unit's range will further halve the damage.

A unit ordered to suppress will cause significantly lower damage to the target, but this can build up suppression within it. When the suppression value of the target is over the suppression threshold, the target cannot move or fire until the suppression value drops back below the threshold.

Ambush mode takes a turn to set up, during which a unit will hide. Damage dealt from hiding occurs first. This can allow hidden units to deal lethal damage to an enemy before they can even return fire.

Terrain Systems

The terrain on the map will be randomly generated with a Perlin noise algorithm. Terrain elements include: plains, forests, swampland, hills, mountains, rivers, and other bodies of water. Some terrain will be difficult for units to traverse, reducing the distance a unit can travel through it. Other terrain will be impossible for units to pass through, unless otherwise specified.

Spotting Systems

The game has fog-of-war by default (can be toggled off in battle rules before starting a new game), meaning that locations of hostile units and buildings owned by enemies are not immediately known to players. Thus, in order to annihilate hostile units or sabotage enemy buildings, a player must spot them. The system can be further broken down into 2 sub-systems: visual and sound, meaning that a unit (the observee) can be spotted either "visually" or by sound by another unit (the observer).

- **Visual:**
  Whether or not the observee is spotted by the observer depends on several attributes of both of the observee and the observer. More details will be provided in the implementation section.
- **Sound:**
  A unit can only be spotted by sound after it has fired in that turn. This is to emulate artillery sound ranging in real-life wars. More details will be provided in the implementation section.

Signaling Systems

In order to assign command to friendly units, they must be receiving signals from friendly cities in that turn. The signal is invisible but can be relayed by other friendly cities or any friendly units. If a friendly unit does not receive any signals, it will become disconnected and the player will not know any details of the unit (e.g. supplies left, strength etc.) until it is reconnected again.

Vehicle Module Systems

All vehicles consist of different modules like cannon breech, turret, fuel tank, engine, track, ammo rack, etc. Each time they receive damage from the enemies, there will be a chance of damaging the modules as well. When the integrity of the

module drops below a certain threshold, the module will have a chance to malfunction every time it is used. When its integrity drops to 0, it must be repaired by Engineers before it can be functional again. For example, the vehicle may misfire if the cannon breech is damaged, and cannot fire at all if its integrity is 0.

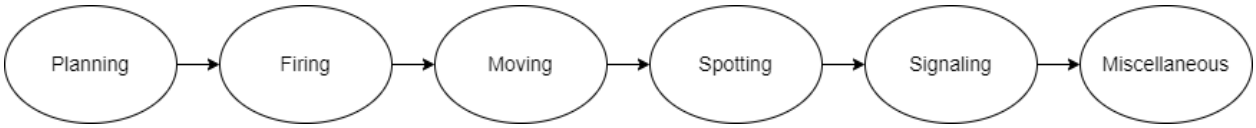
Modules on a vehicle can be replaced for lower cost or higher performance. For example, a larger calibre gun can be mounted on a vehicle in order to increase damage output.

Weather Systems

Every few turns, a season change will occur, causing terrain to have different properties. For example, rivers may freeze and swampland may become extra boggy.

Phases

Each turn is divided into several phases, during which players can only interact with the game during the Planning phase (e.g. giving commands to units, training units etc.) under a time limit (default is 120 seconds, configurable in settings), the other phases are carried out automatically and they serve as execution of commands assigned in Planning Phase, calculations and scene updates. For clarity, the flowchart of a turn is provided below:



Balancing and Modability

All attributes of units (e.g. costs, damage etc.), number of resources added each turn are configurable by editing the corresponding JSON file. We adopt the serialization approach, instead of hardcoding all the attributes, for the ease of balancing and testing. In case we would like to change the value of certain attributes, we can simply modify the JSON file, instead of recompiling the whole game. If some players would like to have their own version of balanced attributes, they can change the files as well, and in multiplayer mode, the values are

synchronized with the server host. All updated values will be loaded dynamically when the game runs.

User Interface (UI)

The user interface is an essential part of the game, as it allows players to interact with the game. Creating an understandable and effective UI is critical to the user experience. Various iterations of the UI will be designed as the project proceeds.

The User Interface is built with Unity UI library. The game is separated into two scenes representing Main Menu and Battle accordingly. The purpose is to avoid conflict of UI Objects in two scenarios and ensure neatness of the developing environment. UI Objects are grouped and set as Child Objects of separate Parent Objects. During navigation, Parent objects are set active or inactive to perform the function of showing or not showing the menu accordingly. Animation of transitions will also be added when needed.

Network

In order to allow for multiplayer games, a multiplayer framework must be established. In this project, Unity MLAPI will be used. Unity MLAPI is a mid-level networking library created for Unity that allows programmers to abstract networking [3].

The multiplayer server needs to be hosted either on a private server or on a cloud server. Both options offer a variety of different benefits and drawbacks. A cloud server has higher availability, but costs around $55 USD a month to run. A private server will have easier setup and maintenance with no fixed monthly cost to run, but to set up a private server may require a higher starting cost, as well as needing to jump through some internet port forwarding hoops. With this in mind, the server will be run on a private device for now, and will possibly be moved to a cloud server depending on our needs.

Bot Algorithm

In order to create a challenging single player historic mode, a well-developed bot algorithm is required. A responsive and adaptive non-player character which behaves similar to humans will be progressively designed based on the necessity and scenario of the game. The behavior will be adjusted corresponding with the amount of resources generated and logistics collected every turn; the weather and terrain of the battlefield; and if an enemy has been spotted during the game.

2.2. Implementation

The project is divided into 3 parts for the ease of implementation. They are assets, game mechanisms and integration. Assets include all resources related to our game, namely modules of tiles and units, music and sound effects, and User Interface designs. Game mechanisms include, but are not limited to, all systems mentioned in part 2.1. Integration oversees the merging of the above parts.

Assets

All modules in the game are self-drawn. We use Blender to create simple animations of the models. Blender is a free and open-source software for 3D model creation. It provides functions like modeling, rigging, and animation.

Royalty-free sound effects are also used. For creation of background music, Cakewalk is used. Design of a single track of background music is planned, but more can be designed if time allows.

FULLY Implemented

Models for terrain tiles are drawn. Adding colliders for terrain is not planned, that is, there will be no collisions between units and terrains, because the sizes of terrain models are significantly smaller (less than half the height) than those of units. So that units can pass through the terrain models without being visually obstructed. Depending on the time schedule, collisions of some terrain models like mountains can still be added, so that it looks more natural when units pass through the terrain.

PARTIALLY Implemented

Some models for vehicles are drawn according to blueprints found online (see 7.1). They provide front views, top views and side views of the vehicles. In blender, the models can be created by starting from the default cube, applying different techniques such as extruding and transforming for editing shapes of the model, boolean modifier for combining / subtracting two different meshes (e.g. combining the turret and the hull of a tank model), bevels for smoothening sharp edges and corners etc.

TO BE Implemented

Models for personnel are yet to be drawn. We planned to use models of soldiers to represent personnel units. However, the difficulty of creating human models from scratch is much higher than that of vehicles. As such. simpler models like military helmets will instead be used for the representation. Animations and texture of models have yet to be created.

Game mechanisms

Game mechanisms are mainly logic and rules that can be represented by code. As our game is coded in C#, Visual Studio is chosen as the Integrated Development Environment (IDE) for code compilation and development. For clarity, all items highlighted in red in the flowcharts are attributes. All formulas involved are listed in Section 7.3.

FULLY Implemented

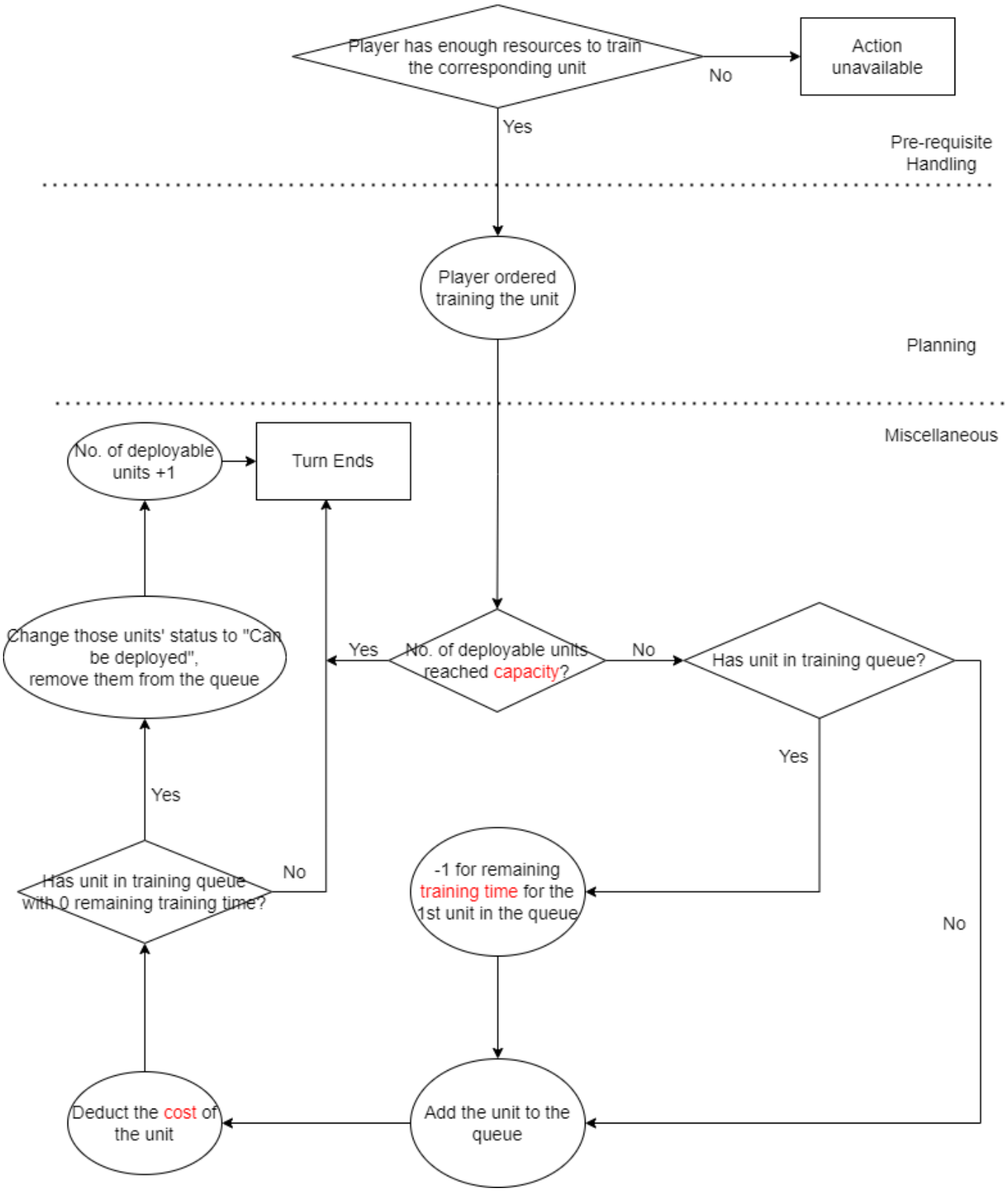
- Resources System
  The implementation of the Resources System was rather simple: If the player is not defeated yet, add the amount of resources to the player. This is carried out in the miscellaneous phase.

- Terrain System
  The Terrain System consists of different modifiers which are taken into consideration during different phases (e.g. moving and spotting). Implementation was simply assigning these values.

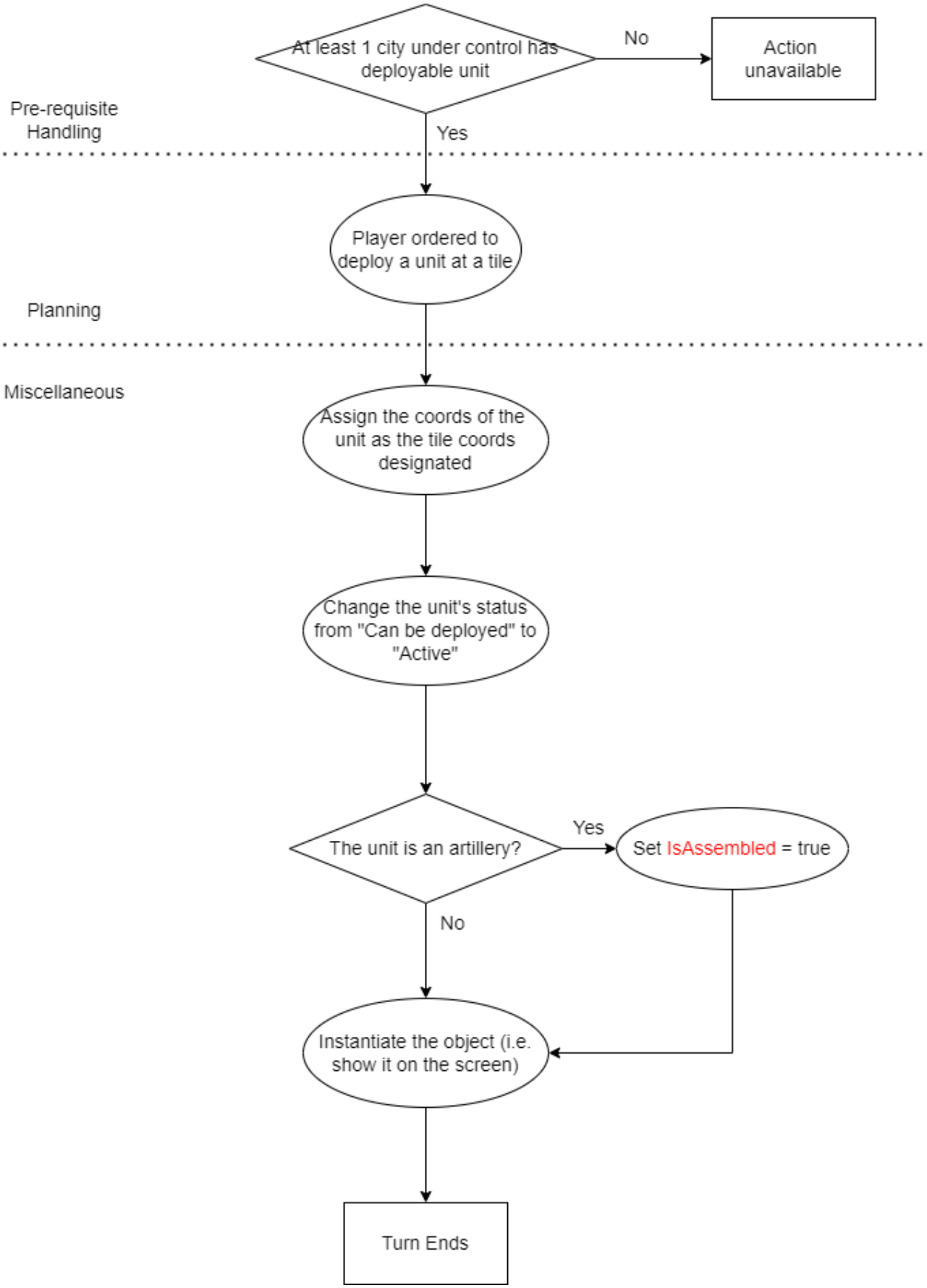
- Unit Creation System

  Units are trained within cities and it takes time to complete training. They can be deployed within 2 tiles around the city. There are several categories of units and units of the same category share similar properties (e.g. vehicles move faster than personnel in general). Only one unit of the same category can be trained at the same time. If extras are trained, they will be put into the queue. The implementation was as follows:

- Train:


Player has enough resources to train the corresponding unit
No
Action unavailable
Yes
Pre-requisite Handling
Player ordered training the unit
Planning
Miscellaneous
No. of deployable units +1
Turn Ends
Change those units' status to "Can be deployed", remove them from the queue
Yes
No. of deployable units reached capacity?
No
Has unit in training queue?
Yes
Yes
Has unit in training queue with 0 remaining training time?
No
-1 for remaining training time for the 1st unit in the queue
No
Deduct the cost of the unit
Add the unit to the queue

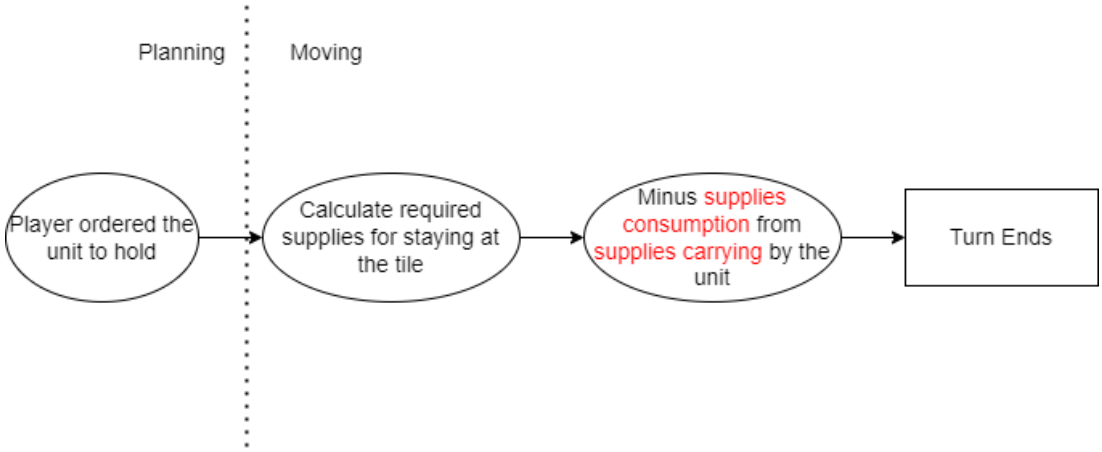
- Deploy:


At least 1 city under control has deployable unit
No
Action unavailable
Pre-requisite Handling
Yes
Player ordered to deploy a unit at a tile
Planning
Miscellaneous
Assign the coords of the unit as the tile coords designated
Change the unit's status from "Can be deployed" to "Active"
The unit is an artillery?
Yes
Set IsAssembled = true
No
Instantiate the object (i.e. show it on the screen)
Turn Ends

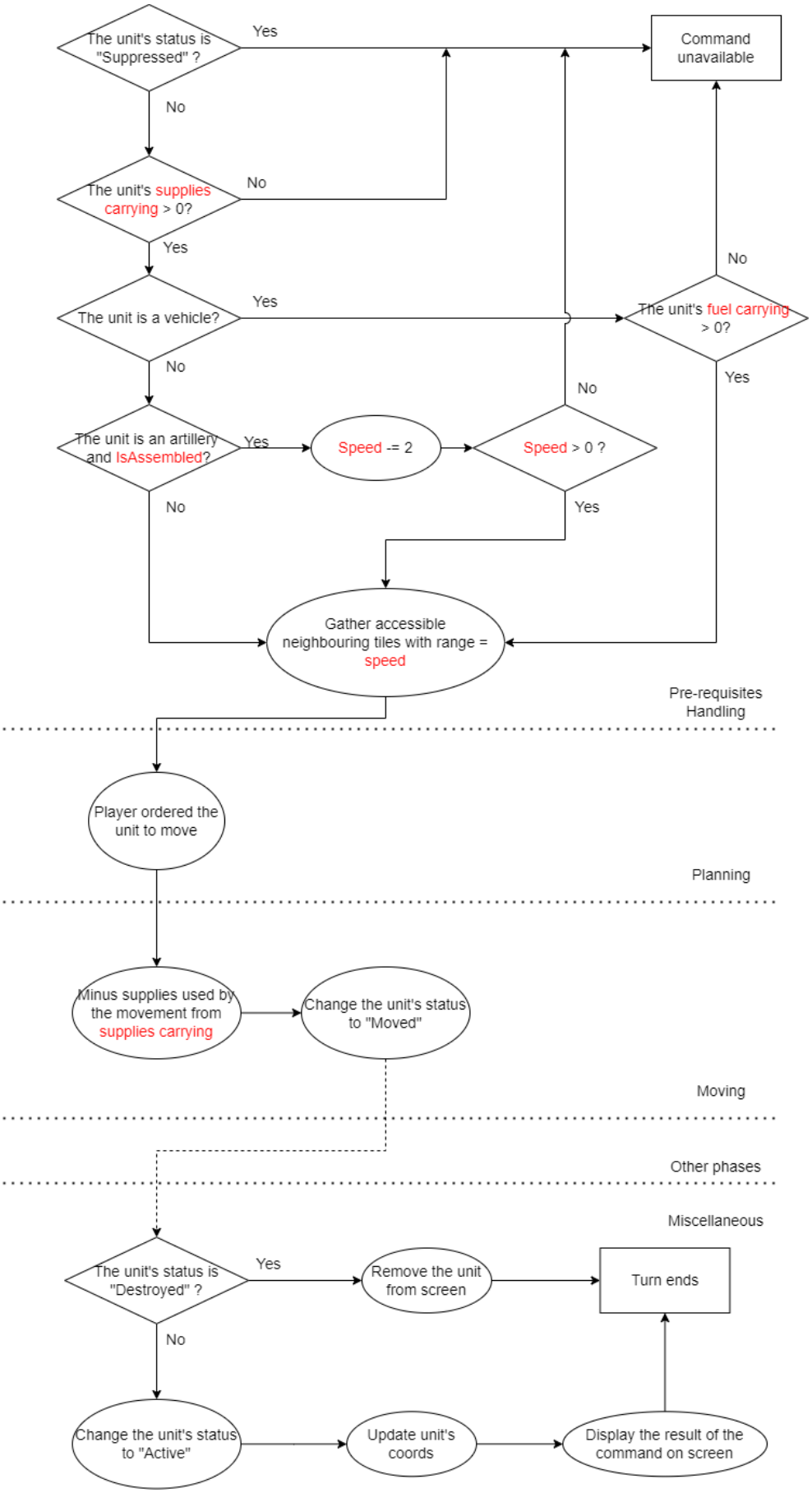
- Combat System

  The combat system consists of several commands, and their implementation were as follows:

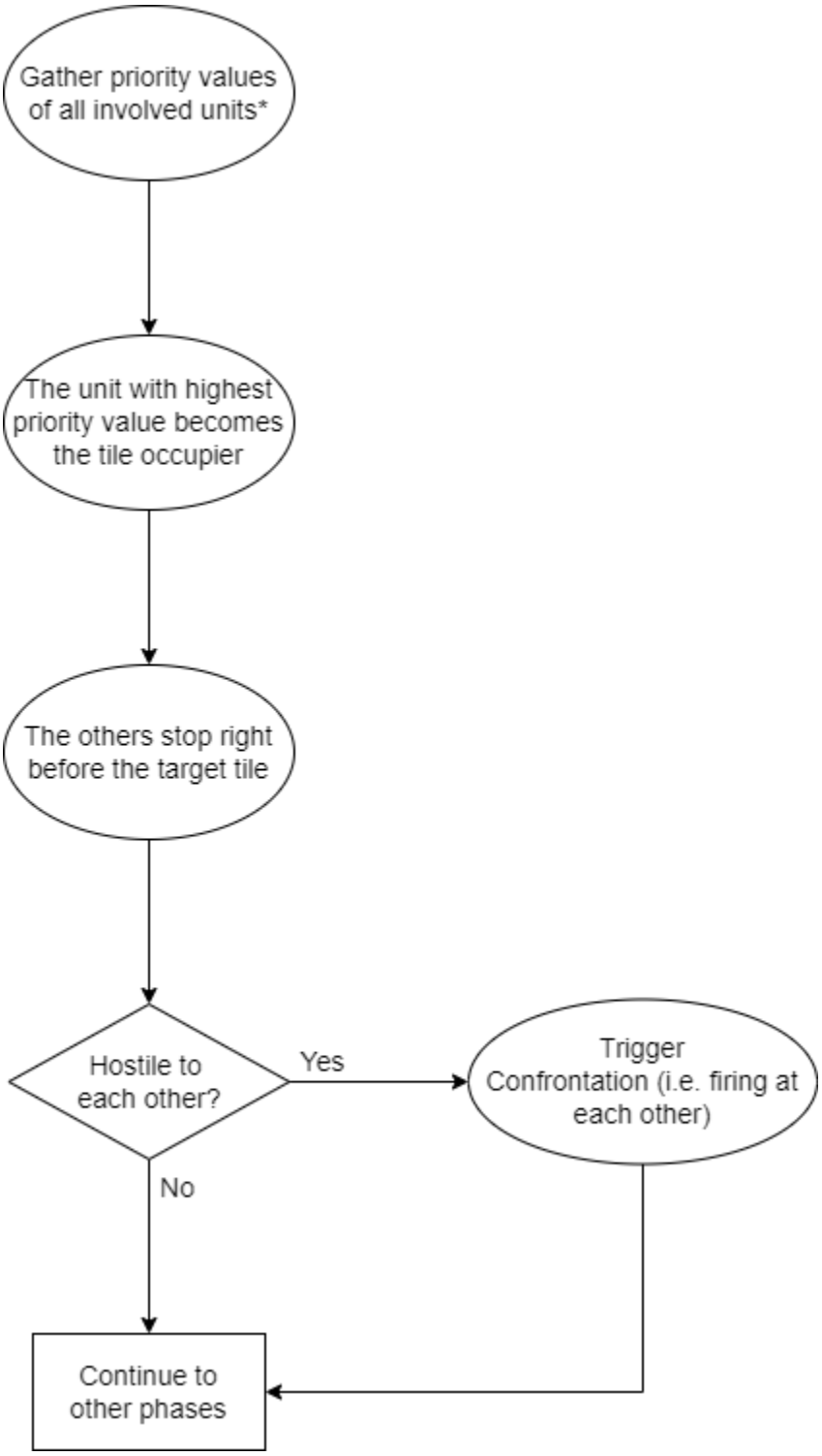
  - Hold:



- Move:

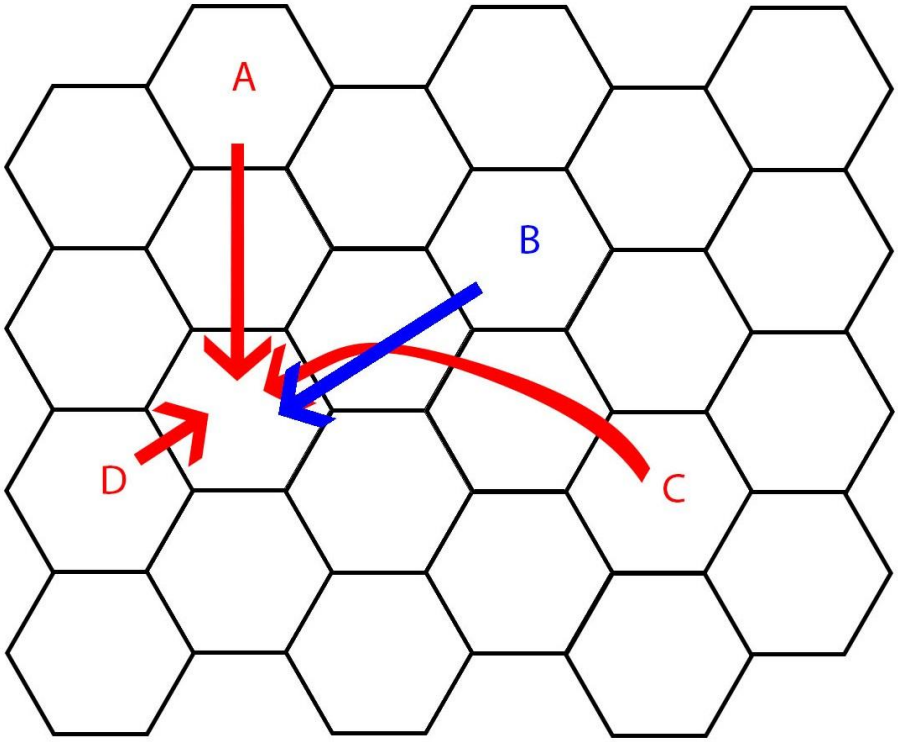

The unit's status is "Suppressed" ?
Yes
Command unavailable
No
The unit's supplies carrying > 0?
No
Yes
The unit is a vehicle?
Yes
The unit's fuel carrying > 0?
No
Yes
No
The unit is an artillery and IsAssembled?
Yes
Speed -= 2
Speed > 0 ?
No
Yes
Gather accessible neighbouring tiles with range = speed
Pre-requisites Handling
Player ordered the unit to move
Planning
Minus supplies used by the movement from supplies carrying
Change the unit's status to "Moved"
Moving
Other phases
Miscellaneous
The unit's status is "Destroyed" ?
Yes
Remove the unit from screen
Turn ends
No
Change the unit's status to "Active"
Update unit's coords
Display the result of the command on screen

- Resolving move conflicts:
  As the game adopts simultaneous turns, there will be some occasions that two or more units, be them friendly to each other or not, are ordered to move to the same tile (target tile). The solution is implemented by the following logic:

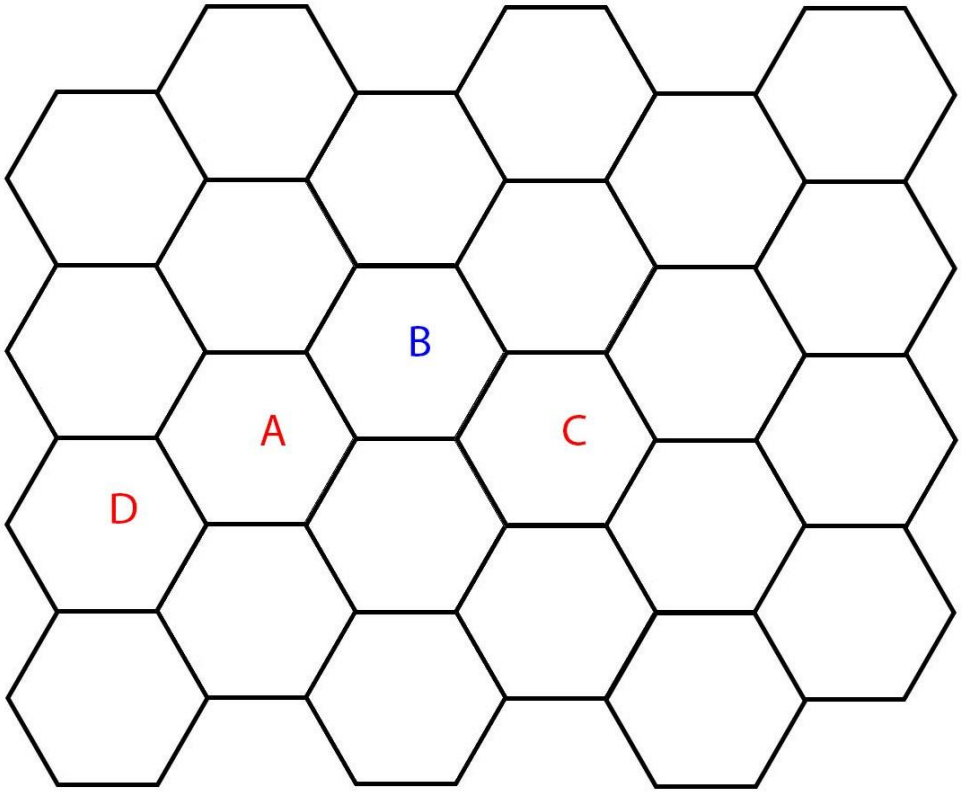


The priority values are determined by a simple formula. For clearer explanation, the following illustrations are provided:

Say we have 4 units, A, B, C and D, where A, C and D are friendly to each other, and B is hostile to them, and they are all commanded to move to the same tile by their owners. Before the executions of commands, it looks like the following:



Given that priority values of A > B > C > D. After executions of the move commands, it looks like the following:



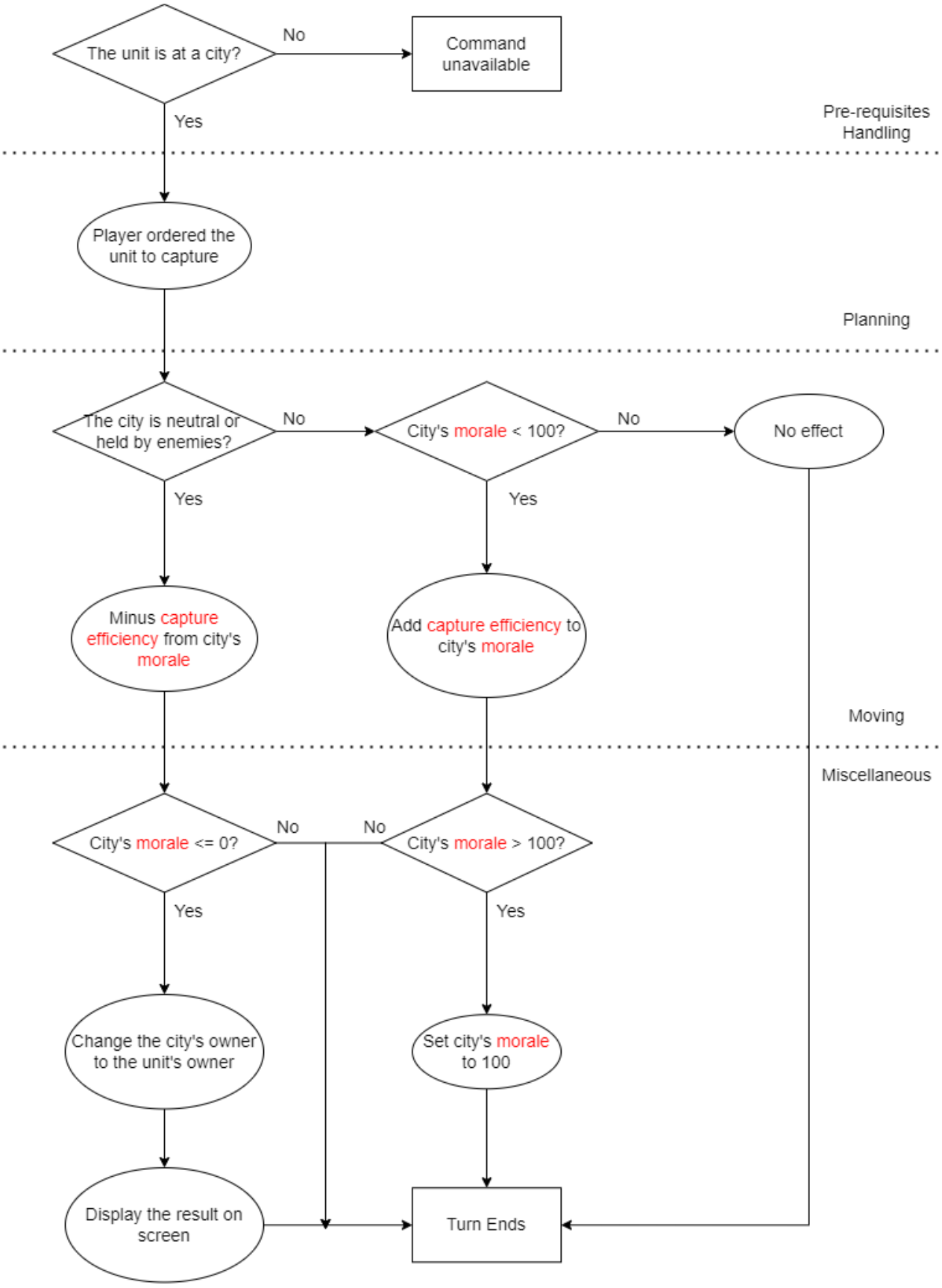
Explanation (movement):

- A has the largest priority value, so it becomes the tile occupier of the target tile.
- Both B and C are going to stop at their second-last tile of their movement path, but that leads to another conflict.
- As priority value of B is higher than C, B takes that tile, and C stops at the third-last tile of its path.
- As priority value of D is smaller than that of A and its movement path only consist of the target tile, it does not move at all.

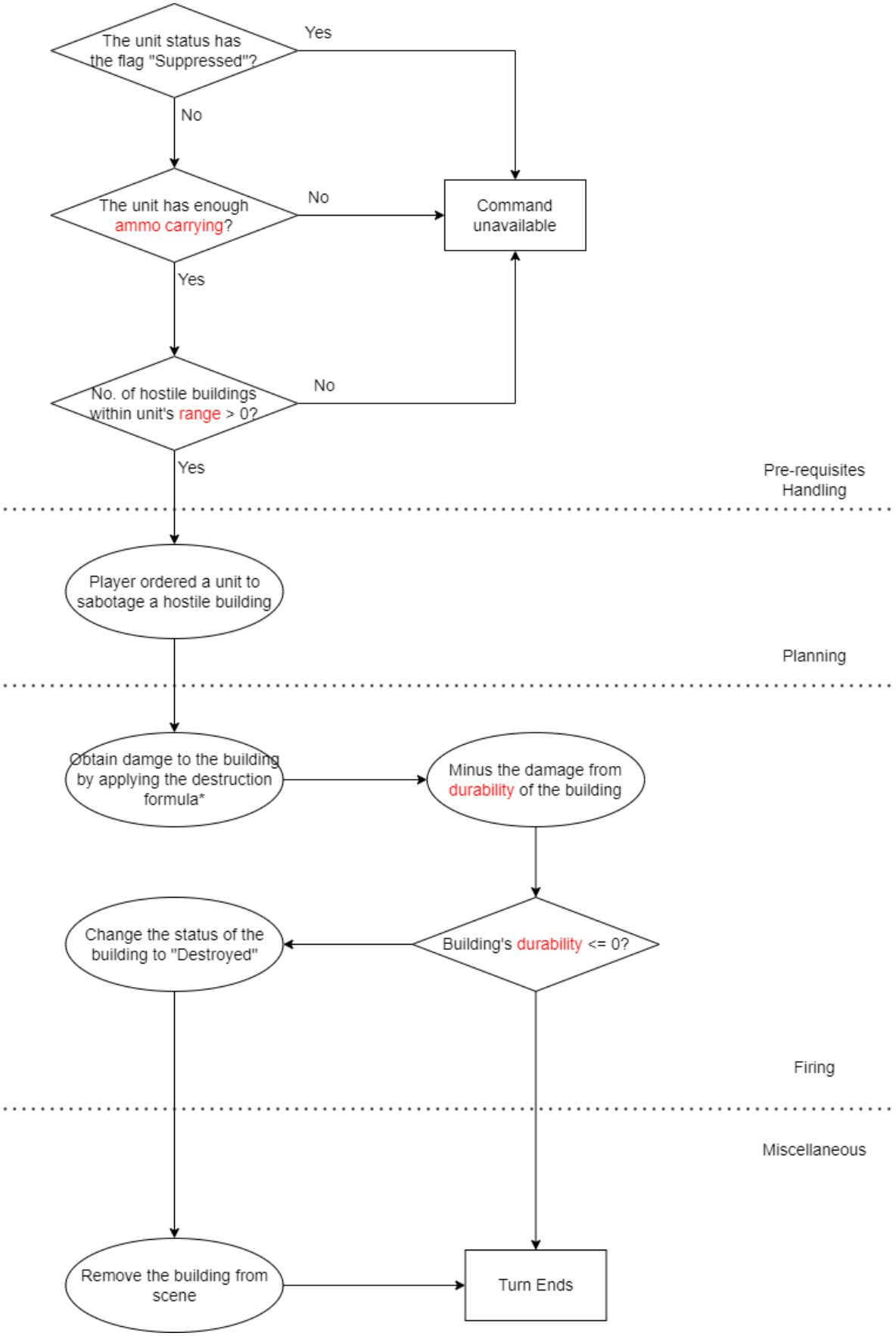
Explanation (confrontations):

- Two confrontations will occur, A vs B and B vs C, as B is hostile to both A and C, if B cannot survive these two confrontations, its status will be changed to “Destroyed” and be removed from screen in the miscellaneous phase.

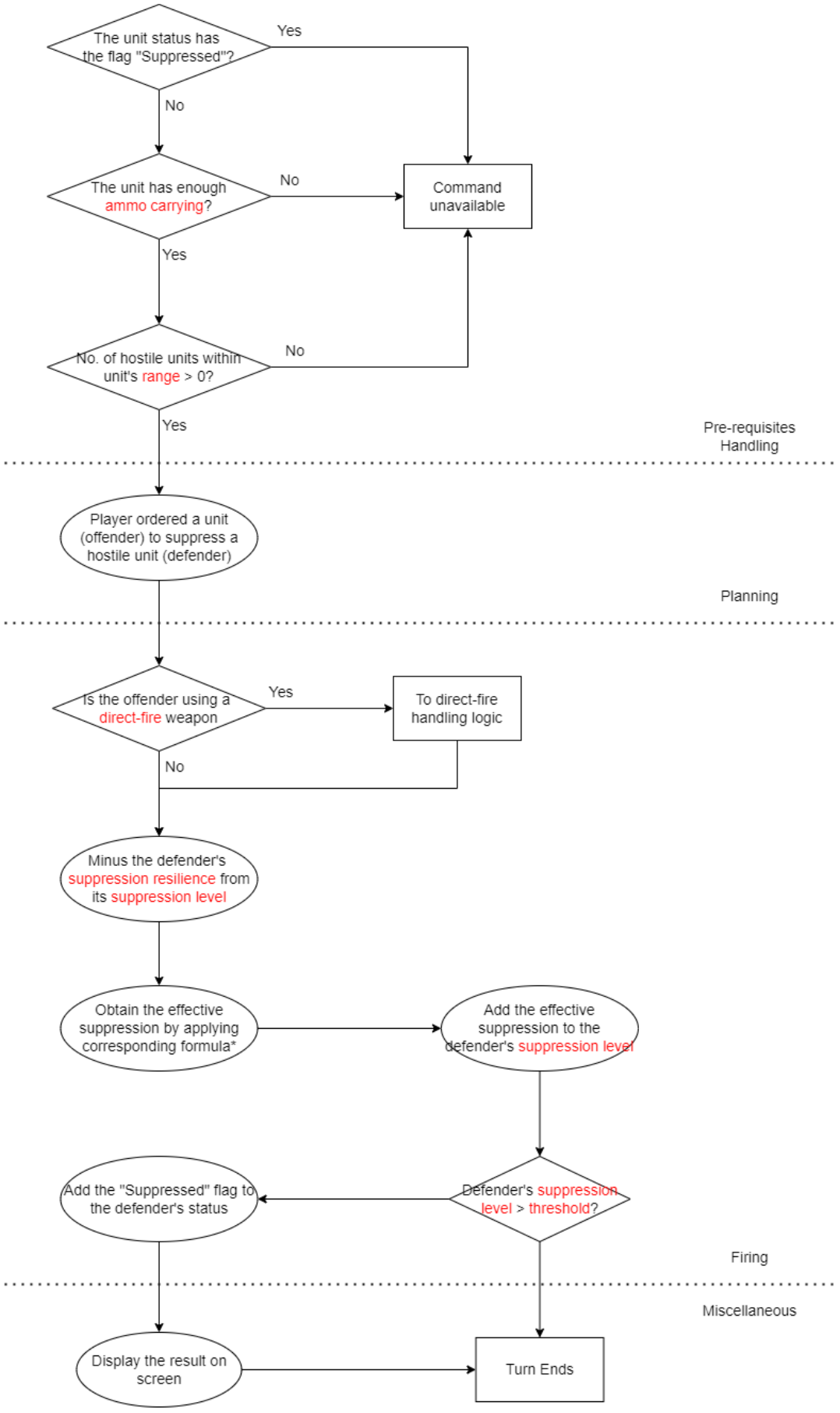
- Capture:


The unit is at a city?
No
Command unavailable
Yes
Pre-requisites Handling
Player ordered the unit to capture
Planning
The city is neutral or held by enemies?
No
City's morale < 100?
No
No effect
Yes
Yes
Minus capture efficiency from city's morale
Add capture efficiency to city's morale
Moving
Miscellaneous
City's morale <= 0?
No
No
City's morale > 100?
Yes
Yes
Change the city's owner to the unit's owner
Set city's morale to 100
Display the result on screen
Turn Ends

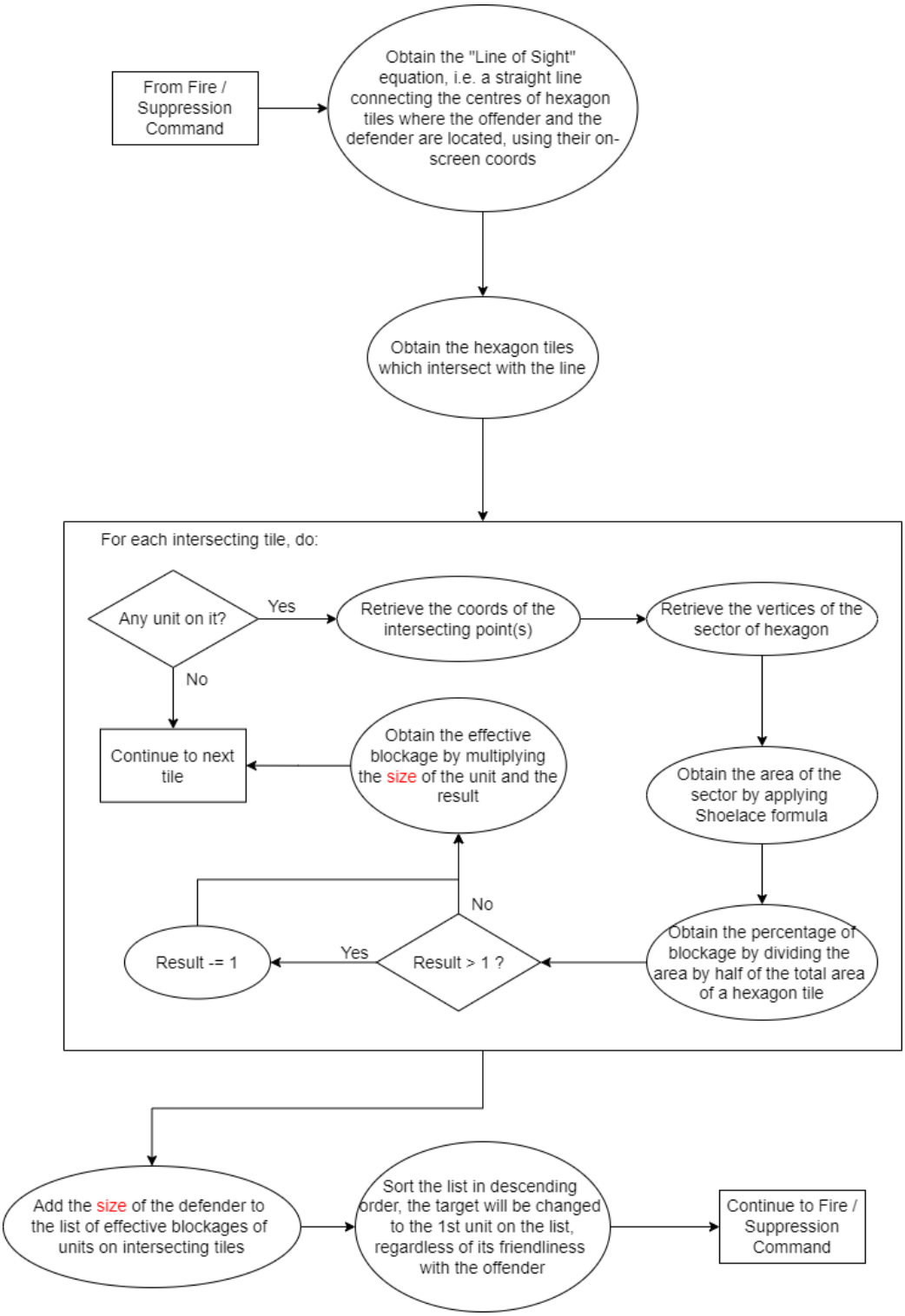
- Fire (See 7.2)
- Sabotage:



- Suppress:


The unit status has the flag "Suppressed"?
Yes
No
The unit has enough ammo carrying?
No
Command unavailable
Yes
No. of hostile units within unit's range > 0?
No
Yes
Pre-requisites Handling
Player ordered a unit (offender) to suppress a hostile unit (defender)
Planning
Is the offender using a direct-fire weapon
Yes
To direct-fire handling logic
No
Minus the defender's suppression resilience from its suppression level
Obtain the effective suppression by applying corresponding formula*
Add the effective suppression to the defender's suppression level
Add the "Suppressed" flag to the defender's status
Defender's suppression level > threshold?
Firing
Miscellaneous
Display the result on screen
Turn Ends

- Direct-fire handling logic:
  Direct fire refers to firing a weapon whose projectile does not travel in a curved ballistic trajectory, in contrast to indirect fire. In our game, friendly fire is disabled, but on some occasions, when the offender is using a direct-fire weapon, it may hit another friendly unit, depending on their positions and sizes.



- Pathfinding algorithm

  The pathfinding algorithm is the implementation of the weighted A* algorithm, where the weight of each node is the supplies/fuel required (depending on optimization option) required for traversing that tile.

PARTIALLY Implemented

- Random Map generation algorithm

  The algorithm can be further broken down into 3 parts:

  ➢ Terrain generation (excluding rivers and cities)

  ➢ River generation

  ➢ Cities generation

  The algorithm for terrain and river generation has been completed.

  The terrain is generated by 2 “maps” created using Perlin noise. One is for “height” and the other is for “humidity”. Although these two items aren’t attributes of terrain, they serve as determinants for the terrain type of each tile. That is, for example, if the tile is with low “height” value (< 0.4) and high “humidity” value (> 0.9), it will become a swamp tile.

  In the original design of the algorithm, river generation was omitted, and river terrains were generated solely using the same logic as used in terrain generation. Yet, we found that rivers would look like ponds instead. So, we modified the algorithm, and took a reference from how the famous game, Minecraft, generates rivers. Firstly, take a tile with high “height” value (>0.9) as the source of the river. Starting from that tile, detect any neighbouring tiles with lower “height” value, extend the river to that tile, add 1 to the counter. Repeat this step until the counter reaches the target length of the river (a random number between ¼ to ½ of diagonal length of the map).

  Right now, the cities generation part of the algorithm is implemented as follows: Randomly pick one tile with at least 10 neighbouring tiles which buildings can be built on, within distance of 2 tiles, as the capital city of one of the players. The next capital city must be at least 1/(total number of players) of diagonal length of the map away from the first capital city. Number of neutral cities equals to 3 * number of players, and they must be at least 10 tiles away from any capital cities. Yet, there is a foreseeable balancing issue with this approach, as some neutral cities may be spawned in a cluster, closer to one of the player’s capital city. A more dispersed generation of neutral cities should be designed.

- Vehicle Module System

The customization for vehicle modules is completed. The mechanism and formula of module damage is yet to be decided.

- Spotting System

  Visual spotting depends on the following attributes: the observer's reconnaissance, detection, and the observee's concealment, where reconnaissance is the maximum range of visual spotting. If the observer's detection is higher than the observee's concealment, the observee is spotted. Both values of detection and concealment are affected by terrains the units are located at, and the status of the unit (e.g. if the unit had just moved in a turn, it will suffer from a penalty of concealment at that turn). Some terrain like mountains will (partially) obstruct visual spotting. The determination of percentage of obstruction is the same as the one used in direct-fire handling logic. Visual spotting is fully implemented in our game.

  The part yet to be implemented is the sound spotting, which depends on the noise generated by the observee after it has fired, and the recognizing thresholds of the observers. The noise propagates throughout the map and suffers from logarithmic attenuation. When three or more observers are within a recognizable threshold lower than that of the attenuated noise at that tile, an approximate location of the observee (appeared in grey, have small chances to be the exact location) will be known to the observers. If the observee fires without relocating for 3 turns, the observers will be able to spot the observee with exact location. The observee is said to be spotted by sound.

TO BE Implemented

- Weather System

  The weather system will be implemented using a simple random generator. When a weather is randomly generated, it changes the value of a flag parameter behind the game which affects the calculations.

- Logistics System

  The logistics system will be implemented by first creating a toggleable logistics view, in which an overlay on the map will be placed showing 2 things, logistics flow and supply routes, and infrastructure such as unit training buildings. The supply routes will be shown on the tiles of the map, flowing from cities to the frontline. Certain city tiles will also have a special flag that

allows them to build infrastructure, giving that tile more commands. These will all be stored client-side.

- Savegame
  The gamestate will be saved in a named JSON file, with which a player can go back and replay from any point that they saved from.
- Record & Replay
  Similar to the savegame system, a replay of every match can be generated by keeping the data of every move that a player makes on each turn, as well as the map and weather. Since the game is deterministic, every game will be able to be encompassed entirely by the commands given, the map, and the weather.
- Achievements
  Achievements are to be defined outside the game's parameters where players need to fulfill certain conditions in order to earn it.

Network

PARTIALLY Implemented

To more easily understand the overall flow of how to code a multiplayer game, a very small and simple test game was created to test out MLAPI. Further implementation is needed to merge the multiplayer framework into the main game.

Integration

We use Unity for the development of our game. As a game engine, Unity supports various rendering operations such as texture compression, mipmaps and various mapping methods. It also includes libraries for audio mixing and multiplayer and networking. These features make integration of assets and scripts much easier. In order to facilitate the workflow of development, GitHub is also used for version control.

2.3. Testing

In order to ensure the game's functionality, different tests will be conducted during different phases of the project.

In the development phase, unit tests will be written in C# using an external library, Unity Test Framework (UTF). Unity Test Framework enables writing test code in both Edit Mode and Play Mode in Unity which ensures each module functions properly and reduces the burden of heavy debug work when the project size grows.

After the User Interface and Interactions are developed, another external library, AltUnity Tester, will be used to provide testing on User Interface elements to ensure every visual element functions as designed.

Software tests will be conducted after the game is fully developed. Play testers will be invited to provide evaluation and bug reports for bug fixes to minimize unexpected glitches.

2.4. Evaluation

Based on the objectives, we must ensure that the functionalities of the game are working as expected in order to maintain a good user experience.

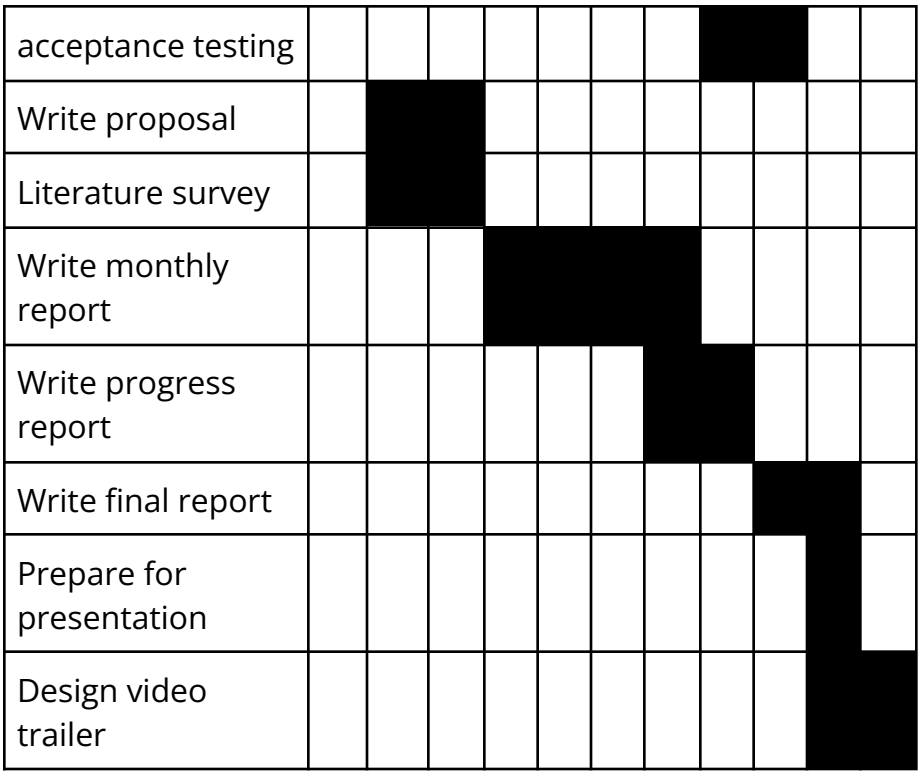
The code will be reviewed in the framework mentioned in 2.3 in order to reduce debug time. We will also review the user interface and design which is one of the important criteria in making the game more enjoyable.

Additionally, user acceptance tests will be conducted to collect feedback from users to make the game more interesting and user friendly. Using the data gathered, analysis on specific components of our project such as the UI, gameplay, and network will allow for feedback-based improvements.

3. Project Planning

3.1. GANTT Chart

Task	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May
Select game engine											
Design game mechanism											
Design Graphical User Interface											
Design game maps											
Design and develop algorithm for bots											
Develop multiplayer mode											
Design background music											
Design and develop map editor											
Design and develop achievement system											
Develop chat function											
Design and develop record & replay function											
Perform unit testing											
Perform user											


acceptance testing
Write proposal
Literature survey
Write monthly report
Write progress report
Write final report
Prepare for presentation
Design video trailer

3.2. Division of work

Task	LEUNG, Ho Man Max	Marcus	Nathan	Hubert
Select game engine	**L**			
Design game mechanism	**L**			
Design Graphical User Interface		**L**		
Design game maps	**L**			
Design and develop algorithm for bots				**L**
Develop multiplayer mode			**L**	
Design background music		**L**		
Design and develop map editor	**L**			
Design and develop achievement system				**L**
Develop chat function			**L**	
Design and develop record & replay function		**L**		
Perform unit testing	**L**			
Perform user acceptance testing	**L**			
Write proposal			**L**	
Literature survey			**L**	
Write monthly report			**L**	
Write progress report			**L**	
Write final report			**L**	
Prepare for presentation	**L**			
Design video trailer		**L**		

*L = Leader, everyone else assisting

4. Hardware and Software Requirements

4.1. Hardware

Hardware	Requirement
Personal Computer	Operating System: Windows 7 (64-bit) or higher

4.2. Software

Software	Version	Usage
Unity	2020.3.17f1 or later	Game development
Visual Studio 2019	16.11.3 or later	Code development
Blender	2.93 LTS or later	Models design
Cakewalk	Sonar Platinum	BGM design

5. References

[1] C. Kelly, P. Johnson, and S. Schuler, “The new face of gaming,” *Accenture*, 27-Apr-2021. [Online]. Available: https://www.accenture.com/us-en/insights/software-platforms/gaming-the-next-super-platform. [Accessed: 12-Sep-2021].

[2] A. Stern and T. Murray, “Top 10 Esports games of 2020 by total Winnings – archive - the Esports Observer,” *The Esports Observer*, 03-Jan-2021. [Online]. Available: https://archive.esportsobserver.com/top10-games-2020-total-winnings/. [Accessed: 12-Sep-2021].

[3] Unity, “Getting started WITH MLAPI: Unity multiplayer networking,” *Unity*. [Online]. Available: https://docs-multiplayer.unity3d.com/docs/getting-started/about-mlapi/index.html. [Accessed: 17-Sep-2021].

6. Appendix A: Meeting Minutes

6.1. Minutes of the 1st Meeting

Meeting 1 - Sept 4th, 2021

Attending:

- Nathan
- Marcus
- Max
- Hubert

- Today mainly do Division of Labor (3.2)
- Asterisks in 3.2 mean optional features
- Props:
    - Free assets? Make our own? Buy assets?
    - For now use default/free assets
- BGM:
    - Marcus will try to do
- Multiplayer server & network
    - Agreement that it would be very preferable that multiplayer is implemented
- Map
    - Perlin noise algorithm
- Achievements
    - Yes
- Chat function
    - Save till last
- Record & Replay
    - Yes

Development Schedule:

- Testing/Rebalancing/Bug fixes: Early March 2021

Use Max's Server

To do:
GANTT chart, table for 3.2, 4.1, 4.2

Flowchart of game

Tentative date of next meeting: Sept 18th, 2021 9pm

6.2. Minutes of the 2nd Meeting

Meeting 2 - Sept 8th, 2021 with Prof Arya

Attending:

- Max
- Marcus
- Nathan
- Prof Arya

Main Questions regarding:

- Objectives
    - What sort of game we're developing
    - How will it be different (**unique features**)
    - Motivations for developing the game
    - Inspiration from different games, in what way we're inspired and what features we want to include
- Lit Survey
    - Comparing similar products
    - Comparing weaknesses and strengths
    - Go into more detail
    - The more we explore the better
    - its flexible
    - Can read both papers and look at other products
- Methodology

- Make sure the game is interesting and fun to play
- But also needs to be a little technical, uses graphics, ai, collision detection etc. (something more advanced, requires original thought)

Fog of war

Maybe group units to move them

Researching during a battle may not seem historically accurate

Historical mode - upgrade during battle

Multiplayer mode - certain number of points to put into upgrades, cannot change during match

6.3. Minutes of the 3rd Meeting

Meeting 3 - Sept 13rd, 2021

Attending:

- Max
- Hubert
- Marcus
- Nathan

Things discussed:

- Game Mechanics
    - How to win: Destroy all enemy bases (starting capital + cities/villages)
    - Logistics
        - Has seperate layer (view)
        - Supplied from outposts
        - Player chooses roads (draws them)
        - Going over different terrain makes it slower (3,2,1)
        - Enemy routes are hidden
        - Killing convoys lets you restock

        - Logistics units running around
        - Can build infrastructure for transport units
            - Roads (trucks)
            - Rails (trains)
        - Maybe have map generate some road parts
        - Maybe logistics units can go over rough terrain, some supply lost, takes longer
    - Unit Resource Consumption (supply)
        - Resource consumption based on tiles travelled, but even not moving takes supply
        - If a unit is on a transport, it does not consume supply
    - Resources
        - Occurs when a player owns Capitals/Cities/Villages
        - Resources added each turn
        - Can construct resource buildings to increase production (marginal return is less)
        - Types of resources

            - Ammo: Produced from munitions factory
                - Can be later split into cartridges/shells
            - (discuss later)
    - Weather system
        - Seasons
        - Terrain affected (eg. rivers freezing over, muddy swamps making it harder to travel)
- Overall flow of the game
    - Start with a capital
    - (discuss later)
- Combat
    - Each unit can either move, fire, or hide(ambush mode)
    - Ambush mode
        - Takes a turn to set up
        - When dealing damage, you deal damage first
        - Moving out of ambush your range is halved
    - If a unit has range 2, and is shooting at a moving target:
        - If the moving target is moving to a tile still in range, damage will be lessened (25%?)
        - If the moving target is moving to a tile not in range, damage will be further lessened (50%?)
- Objective of Project
- Methodology (Part 2)
    - Design
        - Describe structure of game
        - Game flowchart
    - Implementation
        - Classifying what we're going to do into small parts
        - Explain what we're going to do for each part and what tools
    - Testing
        - Content:
            - Mechanisms to test
        - Self testing
        - Automated tests?
        - Send to friends
    - Evaluation
- Hardware and Software Requirements

Things to do by Wednesday:

Max: Lit Survey (1.3) for the first 3, Implementation (2.2), Hardware and Software (4.1, 4.2) Requirements

Nathan: Finish Overview (1.1), Objectives (1.2), add Starcraft inspiration to 1.3, Design (2.1)

Marcus: Testing (2.3)

Hubert: Evaluation (2.4)

6.4. Minutes of the 4th Meeting

Meeting 4 - Sept 18th, 2021

Attending:

- Max
- Nathan
- Hubert
- Marcus

Class diagram to be sent by Max tomorrow
Game class includes all props

Talk further about game mechanics and how to implement them

Hearts of Iron AI (simple approach):

- How to determine which unit to train?
- Probabilistic actions

Rule-based AI

Possibly have units with abilities?

Timeframe?
Keep management simple first: the battle comes first

Todo:

- Search for assets from Unity assets store
- Next Wednesday Max gives us basic files

6.5. Minutes of the 5th Meeting

Meeting 5 - Oct 17th, 2021

Attending:

- Marcus
- Nathan
- Hubert

Big game class that saves gamestate of each turn
Another interface for each player that indicates what info is shown to that exact player
AI can build a bot on top of that interface

- Meet up soon to finalize
- Marcus works on sample buttons in PS
- Nathan works on multiplayer
- Hubert works on AI
- Max working on initial implementation

6.6.

.

6.7.

.

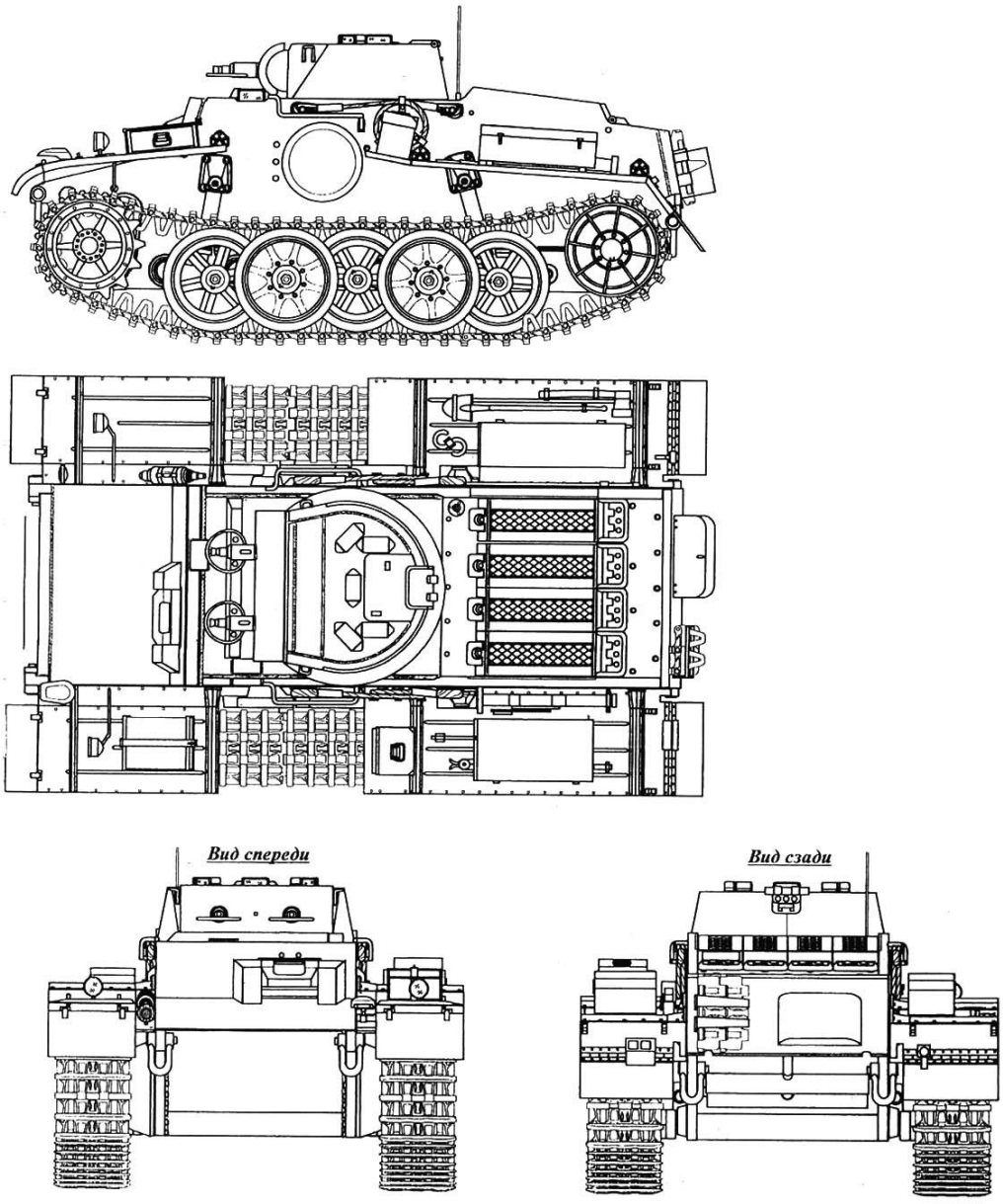
6.8.

.

6.9.

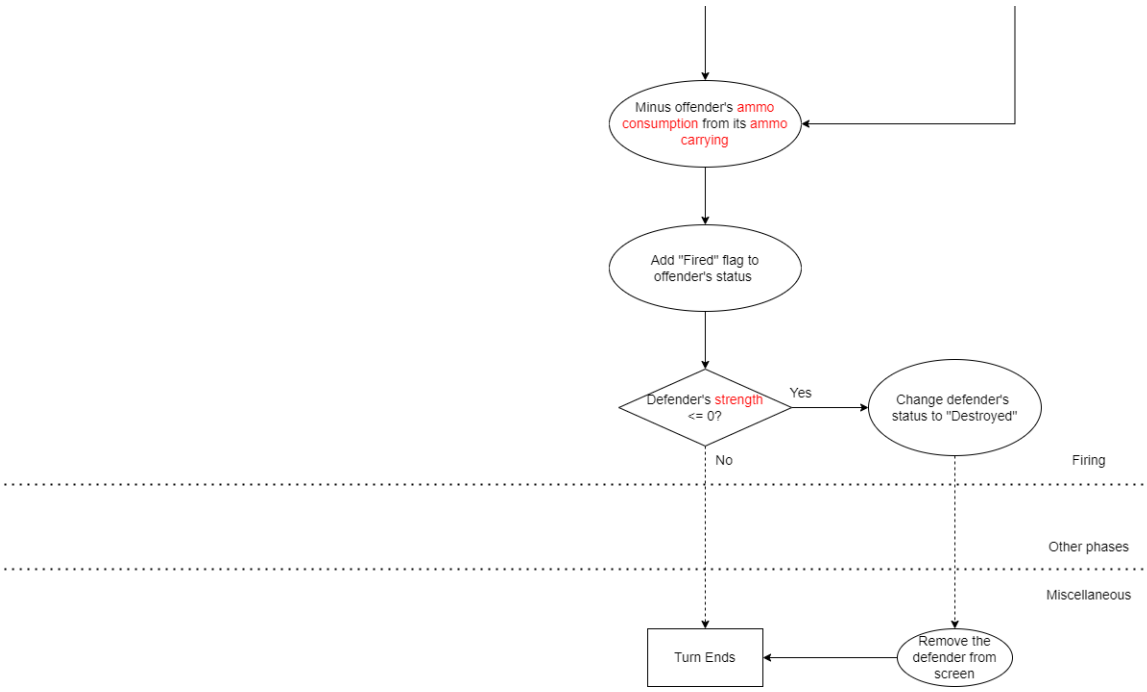
7. Appendix B: Figures

7.1. Example blueprint of vehicle (Panzer I)



Source: https://drawingdatabase.com/panzer-i/

7.2. Fire command logic


Minus offender's ammo consumption from its ammo carrying
Add "Fired" flag to offender's status
Defender's strength <= 0?
Yes
Change defender's status to "Destroyed"
No
Firing
Other phases
Miscellaneous
Turn Ends
Remove the defender from screen

7.3. Formula and functions

For the following formula:

- m is the mobility of the unit.
- mod is the mobility modifier given by the terrain where the unit is located.
- s is the size of the unit.
- w is the weight of the unit.
- $Rnd(x, y)$ means picking a random number from x to y.
- a is the accuracy of the weapon the offender is using.
- σ_a is the deviation of accuracy of the weapon.
- σ_f is the deviation of firepower of the weapon.
- f is the firepower of the weapon.
- e is evasion of the defender.
- h is hardness of the defender.
- $drop$ is the firepower dropoff function depending on d, the distance between the offender and the defender, c, the dropoff coefficient of the weapon, and r, the maximum range of the weapon.
- p is the penetration of the weapon.
- $pendrop$ is the penetration dropoff function, yet to be designed.

Priority value formula:

$priority = \frac{m * mod}{s * w}$

Drop function:

$drop(d, c, r) = \frac{1.4}{\pi} * (cos^{-1}(bcd - 1) - bc\sqrt{- d(d - \frac{2}{bc}}$), and

$b = 1 + \frac{0.6}{r^2}$

Formula for personnel:

$damage = Rnd(a - \sigma_a, a + \sigma_a) * Rnd(1 - \sigma_f, 1 + \sigma_f) * f * (1 - e) * drop(d, c, r)$

Formula for targets without resistance:

$damage = Rnd(1 - \sigma_f, 1 + \sigma_f) * f * 0.25 * drop(d, c, r)$

Formula for weapons without penetration:

$damage = Rnd(a - \sigma_a, a + \sigma_a) * Rnd(1 - \sigma_f, 1 + \sigma_f) * (1 - h) * f * (1 - e) * drop(d, c, r)$

Penetration formula:

$effective\ penetration = Rnd(1 - \sigma_p, 1 + \sigma_p) * p * pendrop$

Formula for penetrated targets:

$$damage = Rnd(1 - \sigma_f, 1 + \sigma_f) * f * drop(d, c, r)$$