THE WAR GAMES

Segmentation Fault

Github Repository: https://github.com/sloththedev/cos214-project

Link to Document:
https://docs.google.com/document/d/1nyAjAcHdjnokLfzVJ3SltEBrp_gzwz5tz_S7Geq3F7s/edit#

Table of Contents

Research

Understanding Warfare:

Land

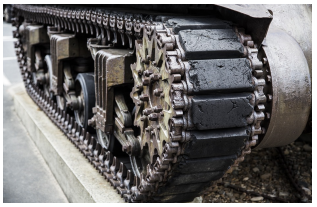
During the 1800s, the main weapons were muskets and bayonets, which is a knife attached to the barrel of a musket (and later on attached to guns). Muskets are single-fire and have poor accuracy. They also took a long time to reload, as the soldier needed to measure the gunpowder, pour it down the barrel of the gun, put a lead ball in and make sure it is sitting on top of the charge powder.

Field artillery, such as mortars and cannons were also used. Cannons are able to fire a single cannonball long distances, causing catastrophic damage. The cannonball would bounce once it hit the floor, allowing it to travel further. If a soldier was unlucky enough to not die on impact, they would end up having a slow, painful death, or with shattered bones, in which the limbs would end up needing to be amputated. Unlike cannons, mortars fired explosive shells and have a shorter range, designed to take-out reinforcements, light vehicles, and troops nearby.

As technology improved, muskets evolved into rifles, which had better accuracy and took less time to reload, due to the formal introduction of bullets. Bullets were designed in a way to be more aerodynamic, reducing drag which decreased bullet drop. The optimal amount of gunpowder was preloaded into the bullet casing, so soldiers no longer had to guess how much gunpowder was needed. Tracer bullets were developed to allow soldiers to see where their fire was directed (which was mostly beneficial at night) and adjust their aim if necessary.

Cavalry units preceded tanks, initially consisting of soldiers fighting on horseback. As machine guns and rifles became more popular on the battlefield, warhorses and their riders were vulnerable, leading to them being withdrawn from the battlefield and needing to be replaced.

Tanks were a new way to move up the frontline with the troops, providing firepower, mobility and protection to standard infantry men. Tanks were multi-crewed with at least one main cannon and multiple supporting machine guns. A key feature of a tank is its continuous tracks that rotate around its axles. This allows for more freedom, unlike a train which is bounded by its tracks.



Tank ammunition included armour piercing, incendiary, high explosive, fragmentation/anti-personnel and during the second world war some were fitted with flamethrowers. In addition, tanks had mini mortar tube-like attachments that fired smoke grenades to conceal their position. Armour piercing rounds were used to destroy enemy tanks, armoured vehicles or fortified positions. Incendiary rounds, designed to ignite enemy vehicles or fuel supplies, causing devastating effects on logistics and supplies. High explosive shells contain an explosive charge, intended to destroy church towers (negating enemy sniper nests) and entrenched positions such as mortar pits. Anti-personnel rounds' main purpose was to eliminate infantry threats as the round itself was filled with deadly shrapnel, inflicting as much bodily harm as possible.

Mechanised troop transport was developed to move troops and field artillery while keeping up with the rapid changes in frontline positions. For example, during the 19th century, troops would only be able to move at most 1 mile a day due to the majority of troops being on foot, in comparison to the 20 miles a day in the Second World War. Initially, it was composed of unarmoured trucks which were then replaced with enclosed, armoured personnel carriers (APCs). These were armed with machine guns to protect the troops as they disembarked and provided support during battle.

Air

Aerial warfare started in the early 20th century, but only became widespread during the Great War (1914 - 1918). Initially, they were unarmed and used to locate enemies. They were then equipped with short range machine guns that only fired in a forward direction and used to shoot down the reconnaissance aircrafts and blimps.

From that, dogfighting was born. Dogfighting is a close-range air duel between fighter planes, and may be referred to as a ‘furball’ if the battle gets intense enough.

Aircraft used in aerial warfare are predominantly categorised into two classes, namely fighters and bombers. Bombers can be classified into three main categories, heavy, medium and light/dive bombers.

Heavy bombers are multi-engined, multi-crewed, heavy payloaded and long range. They also had heavy defensive armaments, in the form of turrets to ward off enemy fighter interceptors. The tactic heavy bombers employed during the day (daylight bombing) was known as carpet bombing, used to bomb air fields, factories, oil refineries and cities.

Medium bombers were mostly used for carpet bombing but on a smaller scale with their main target being bridges, defensive emplacements, and harbours. They had less defensive armaments, as well as a reduced payload.

Light bombers were designed for combat air support (CAS), circling an area waiting on instructions from ground forces and were usually armed with small bombs, machine guns and High Velocity Aerial Rockets (HVAR). They preyed upon targets of opportunity such as tanks, locomotives, and transport.

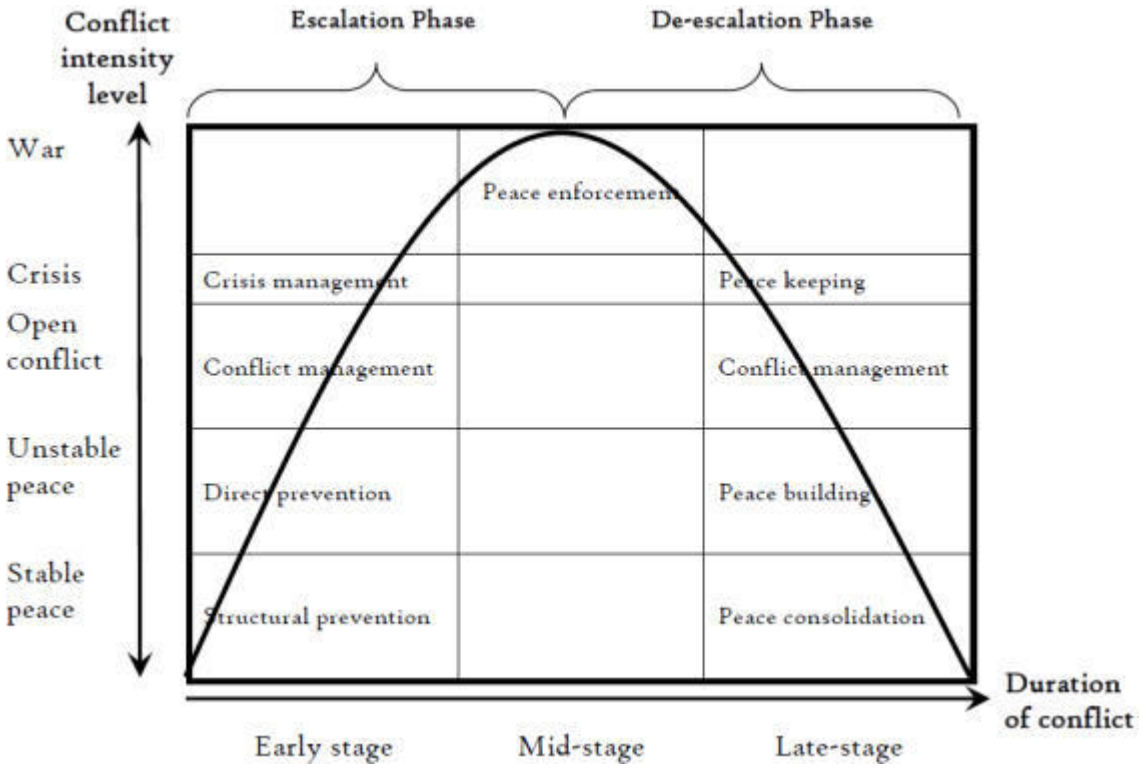
Fighter planes had the following roles: interception, escort, combat air patrol (CAP) and CAS. Interceptors attacked bomber streams while escorts protected the bombers. CAP's main purpose was to patrol an area and ensure that no enemy aircraft entered said area. Fighter aircraft were usually single-crewed, single or twin-engined and were primarily armed with machine guns, cannons, HVAR and bombs.

Phases of War:

Main Phases

The two main phases of war are the escalation phase and the de-escalation phase. The escalation phase comprises of the Early-Stage and part of the Mid-Stage and the de-escalation phase comprises part of the Mid-Stage and the Late-Stage. The transitions between the phases depend on the intensity levels of conflict:

Stable Peace ⇋ Unstable peace ⇋ Open conflict ⇋ Crisis ⇋ War



Visual representation of war phases [12]

Early Stages

The early stages of war are what breaks the peace between two factions, be it countries, political parties, alliances, (or even animals like in the Great Emu War) etc. This list is non-exhaustive. It also may be “immediate” or gradual. For example, in World War 1, it was

immediate as it was a gunshot that triggered the start (there was already mistrust between the countries, however, the assassination of Franz Ferdinand, the Archduke of Austria was the catalyst amongst the growing discourse). World War 2, on the other hand, was gradual as the peace was slowly breaking over a few years before the war was declared. This was also a period of political intimidation and alliance building.

Once the peace has been disturbed, the different factions start to prepare for what may or may not follow. This usually includes training troops, mass manufacturing weapons, ammunition, building of vehicles such as tanks, aircrafts, and mechanised transport. Planning also takes place in this stage, spies are sent to gather information of the enemy faction so they may counteract or attempt to foil the enemies plans. Strategy is essential. Logistics are prepared, such as who and what needs to go where, to set a solid foundation for their position in the battle. Like building bunkers, pillboxes, and forts etc.

Mid Stages

The mid stage is the main stage, where hostility is at its highest and the troops are sent into battle, to fight for their faction. Tactics are the most important element in this stage, it will determine which faction prevails.

The Mid-Stage can be described with the following hierarchy:

Campaign
Operation
Battle
Skirmish

This is a simplification of definitions due to the dynamic nature of warfare, as there exists no constant sizes during a war.

A skirmish is a small fire-fight between two opposing platoons, which has a very short duration lasting a few hours but not more than a day.

A battle is a fire-fight between opposing companies (a company is made up of two or more platoons), fought over a few days to a few weeks.

An operation is fought between multiple opposing battalions (which is made up of two or more companies), ranging from as little as a day to as long as a few months.

A campaign is fought between two opposing brigades covering a vast area and lasting several months to even years. For example, during WW2 the North African campaign stretched from Cairo to Casablanca, and persisted for roughly 2 years.

End Stages

The late stage is when retreating and surrendering are on the table. Troops may choose to surrender when resistance becomes futile, and more would be lost if the battle continues. There are two types of surrender, conditional and unconditional.

Unconditional surrender is when there is no guarantee for anything for the surrendering party, they ask for nothing in return when surrendering. Conditional surrender is when the surrender occurs only when certain demands/conditions are met, otherwise the battle continues.

If soldiers were captured by the enemy, they would be referred to as Prisoners of War (PoWs). In early wars, they were usually executed or enslaved. As time progressed, PoWs became more protected, even more so under the Geneva Convention - PoWs need to be treated humanely, at all times and may not be subjected to violence, intimidation, mutilation or anything that may harm the PoWs in any aspect. At the end of the war, PoWs must be released and repatriated. Repatriation may include money and the rebuilding of cities destroyed during the war. Fallen soldiers were usually repatriated, where countries could collect the bodies and bury them in home soil with full military honour.

Assumptions and Decisions:

- Our simulator caters for war during the 1800s - 2000s
- War consists of two alliances, for our simulation we used the west and east, however the system is modular
- Countries control, create, and merge platoons
- If two platoons from the same alliance occupy the same area, then they merge, their country would be the first platoon in the area.
- Economies would grow proportionate to the amount of areas it occupies capped at 1.5% per round
- Platoons with no ammunition can do minimal damage - we assumed they can pick up a rock. If they have lost their arm, they kick, if they have lost their legs, they spit. If they have lost their head…they're dead.
- If a platoon's morale is low or their health is too low, then they will retreat from a battle if they feel that they cannot win the battle.
- The retreat can only take place if there is an adjacent area that is of the same alliance.
- A platoon has two main weapon strategies, precision and explosives.
- The platoon can switch the strategies mid battle, however if they have a certain amount of ammo per weapon.
- When a platoon uses the precision strategy, they will fire as many shots as there are units or soldiers/the amount of bullets they have to shoot.
- Precision does a large amount of damage to a random unit in the enemy platoon, while explosives deal less damage to a larger spread of units in the enemy platoon.
- Transport routes are created adjacently from areas between two areas of the same alliance.
- Enemy alliances can destroy adjacent transport routes from their area.

- After each round the country will try to replenish, it can only replenish if its economy is strong.
- Replenish will call all the countries factories that they own - Medics, Ammo, Goods.
- Medics increase the health of a random amount of injured units
- Ammo will replenish both precision and explosive ammos
- Goods will increase the moral of a random amount of units with low morals
- Platoons could either be a land or air branch
- Land platoons cannot attack air platoons, however air platoons damage land platoons to a significantly large amount.
- Countries' populations at a fixed rate, and the larger the population the more soldiers will enlist to join platoons. Thus platoons can only grow by a limited number over the course of a few rounds.

We have two interfaces - real and design. Real mode runs a random simulation with a CPU, until the war ends. With design mode, the user creates the scenario and has the opportunity to let the CPU run or still let the user decide.

Definitions:

- Carpet bombing: also known as saturation bombing. Its main purpose is to wreak havoc over a large area, destroying everything within it.

- Gun drop: rate at which a bullet will lose precision due to loss in horizontal velocity.

- HVAR: unguided rocket, nicknamed Holy Moses.

- Sniper: a soldier shooting long range rifles with pristine accuracy, usually hiding in high places.

- Skirmish: an unplanned, light battle, usually between two small opposing groups.

- Tracer bullets: bullets that comprise of a pyrotechnic column at the base that is ignited when the bullet is shot, creating a visible streak which can be used to correct one's aim.

- Fighter Aircraft: a fighter plane's main purpose is to maintain control of airspace by eliminating enemy aircrafts. They are usually lightweight and agile.

- Bomber Aircraft: they have a durable structure which allows it to carry heavy loads, like bombs and rockets, and have defensive armaments. These aircrafts are used to destroy land targets.

- Great War: also known as World War 1

References:

[1] ——, *The Weapons of War* (2014) *Shakespeare Birthplace Trust*. Jamie Weisz. Available at: https://www.shakespeare.org.uk/explore-shakespeare/blogs/weapons-war/#:~:text=Firearms,-In%20Elizabeth%20I's&text=Muskets%20later%20evolved%20into%20rifles,much%20lighter%20and%20more%20portable (Accessed: October 27, 2022).

[2] ——, *Bullet* (2022) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/bullet (Accessed: October 27, 2022).

[3] Bryant, J. (1999) *How to load a musket.*, *Sonsofdewittcolony.org*. Available at: http://www.sonsofdewittcolony.org/adp/history/1836/the_battle/the_weapons/load.html (Accessed: October 27, 2022).

[4] MacIsaac, D. (2016) *Air Warfare*, *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/topic/air-warfare (Accessed: October 27, 2022).

[5] ——, *Fighter aircraft* (2022) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/fighter-aircraft#ref172441 (Accessed: October 27, 2022).

[6] ——, *Cavalry* (2013) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/topic/cavalry (Accessed: October 27, 2022).

[7] Ogorkiewics, R.M. (2022) *Tank*, *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/tank-military-vehicle (Accessed: October 27, 2022).

[8] ——, *Mortar* (2022) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/mortar-weapon (Accessed: October 30, 2022).

[9] ——, *Bayonet* (2015) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/bayonet (Accessed: October 30, 2022).

[10] ——, *Bomber* (2017) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/bomber-aircraft (Accessed: November 1, 2022).

[11] ——, *Strategic bombing* (2016) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/topic/strategic-bombing (Accessed: November 1, 2022).

[12] Aliyu, Aliyu & Kasim, Rozilah & Martin, David & Raja, Parit & Pahat, Batu & Johor, Darul & Correspondence, Malaysia. (2015). Ethno-Religious Violence and Neighbourhood Facilities Provision: Evidence From Jos, Nigeria. Journal of Environment and Earth Science. 5. (Accessed September 29, 2022)

[13] Crosby, F. (2004) *Bombers: An illustrated history of bomber aircraft, their origins and evolution*. London: Lorenz.

[14] Field, J.F. (2014) *D-Day in numbers*. London: Michael O'Mara Books Limited.

[15] Mann, C. (2012) *Great battles of World War II*. Bath: Parragon.

[16] Rolfe, M. (2015) *Bomber boys*. London: Bounty.

[17] History.com Editors (2009) *World War I*, *History.com*. A&E Television Networks. Available at: https://www.history.com/topics/world-war-i/world-war-i-history (Accessed: November 1, 2022).

[18] Cavanaugh, M.L. (2016) *Anti-morale: What causes retreat and surrender?*, *Modern War Institute*. Available at: https://mwi.usma.edu/2014920anti-morale-what-causes-retreat-and-surrender/ (Accessed: November 1, 2022).

[19] ——, *Prisoners of war: What you need to know* (2022) *International Committee of the Red Cross*. International Committee of the Red Cross. Available at: https://www.icrc.org/en/document/prisoners-war-what-you-need-know#:~:text=As%20a%20general%20rule%2C%20POWs,can%20lead%20to%20earlier%20release. (Accessed: November 1, 2022).

[20] ——, *Military unit2020* (2020) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/topic/military-unit (Accessed: November 1, 2022).

[21] ——, *Carpet bombing* (2016) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/topic/carpet-bombing (Accessed: November 1, 2022).

[22] ——, *Rocket and missile system* (no date) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/rocket-and-missile-system (Accessed: November 1, 2022).

[23] ——, *Skirmish definition & meaning* (no date) *Merriam-Webster*. Merriam-Webster. Available at: https://www.merriam-webster.com/dictionary/skirmish (Accessed: November 1, 2022).

[24] ——, *Military aircraft* (2018) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/technology/military-aircraft (Accessed: November 1, 2022).

Image References:

[25] *Napoleonic reenactment historical - free photo on Pixabay* (no date) *Pixabay*. Available at: https://pixabay.com/photos/napoleonic-reenactment-historical-1712674/ (Accessed: November 1, 2022). [Musket]

[26] *Gettysburg Pennsylvania - free photo on Pixabay* (no date) *Pixabay*. Available at: https://pixabay.com/photos/gettysburg-pennsylvania-battlefield-350058/ (Accessed: November 1, 2022). [Cannon]

[27] *R/funny - anatomy of a Pew* (no date) *reddit*. Available at: https://www.reddit.com/r/funny/comments/7l8xr8/anatomy_of_a_pew/ (Accessed: November 1, 2022). [Anatomy of a Pew]

[28] *Tracer Bullets* (no date). Available at: https://commons.wikimedia.org/wiki/File:Tracer_rounds_light_up_the_sky_as_a_nighttime_live-fire_exercise_takes_place_during_the_joint_US-South_Korean_Exercise_TEAM_SPIRIT_%2784._Member

s_of_the_8th_Security_Police_Squadron_A_-_DPLA_-_af41b3cd79d662157c163538ee4c4066.jpeg (Accessed: November 1, 2022). [Tracer Bullets]

[29] Afd (no date) *Cavalry Army reenactment photos in .JPG format free and Easy Download Unlimit ID:273330*. Available at: https://all-free-download.com/free-photos/download/cavalry_army_reenactment_273330.html (Accessed: November 1, 2022). [Cavalry]

[30] *Tank War Armour - free photo on Pixabay* (no date) *Pixabay*. Available at: https://pixabay.com/photos/tank-war-armour-heavy-vehicle-203496/ (Accessed: November 1, 2022). [Tank Tracks]

[31] *Tank War battlefield - free photo on Pixabay* (no date) *Pixabay*. Available at: https://pixabay.com/photos/tank-war-battlefield-army-shells-449772/ (Accessed: November 1, 2022). [Tank]

[32] *Military Vehicle World War 2 Arm - free photo on Pixabay* (no date) *Pixabay*. Available at: https://pixabay.com/photos/military-vehicle-world-war-2-army-4077568/ (Accessed: November 1, 2022). [Transport]

[33] *HD wallpaper: Brown war aircraft, fighter, art, airplane, painting, aviation* (no date) *HD wallpaper: brown war aircraft, fighter, art, airplane, painting, aviation | Wallpaper Flare*. Available at: https://www.wallpaperflare.com/brown-war-aircraft-fighter-art-airplane-painting-aviation-wallpaper-qgzvp (Accessed: November 1, 2022). [Dogfight]

[34] Karol, M. (2019) *B-17 squadrons bombing Rome*, *World War Wings*. Available at: https://worldwarwings.com/b-17-squadrons-bombing-rome/ (Accessed: November 1, 2022). [Heavy Bomber]

[35] *American, B 25, bomber, flight, medium, military, Mitchell, north, Sky, HD wallpaper* (no date) *Wallpaperbetter*. Available at: https://www.wallpaperbetter.com/en/hd-wallpaper-gggkc (Accessed: November 1, 2022). [Medium Bomber]

[36] *HD wallpaper: De havill mosquito, Light Bomber, World War Two, De Havilland* (no date) *HD wallpaper: De Havill Mosquito, light bomber, world war two, de havilland* | *Wallpaper Flare*. Available at: https://www.wallpaperflare.com/de-havill-mosquito-light-bomber-world-war-two-de-havilland-wallpaper-tvxek (Accessed: November 1, 2022). [Light Bomber]

[37] *Fighter Plane* (no date). Available at: https://i.pinimg.com/originals/6c/95/38/6c953881d94b416f8f11661b3d42640c.jpg (Accessed: November 1, 2022). [Fighter Plane]

Application of Design Patterns

Refer to last page for UML

State Design Pattern:

The State Design Pattern has been applied in order to make Players modular. This allows pluggable Players that can have diverse strategies.
The State Design Pattern has also been applied in order to keep track of both individual units (including humans, vehicles and platoons) as well as countries' involvement in the war.

Player State Pattern

The Player Hierarchy employs the state design pattern in order to manage running the war simulator in user mode, GUI input mode or simulation mode. In addition, the user can toggle between these modes many times during a simulation. In order to reduce the complexity of the management of the player, its state is controlled by the God Of War from which users can iterate through states at will, at any time. By externalising the state, the correct functions based on the state are automatically linked at run-time. The different behaviours of the layer are wrapped in a stable interface so that no dynamic parsing is necessary. The concrete states add extra and diverse functionality without impacting the rest of the interaction of the system with it. This allows one to plug in different players at will as long as it conforms to the interface, resulting in increased scalability of the system. One could develop players with different motivations based on stakeholder needs, including players focused on economic gain, aggressive players and defensive players to deliver a more accurate simulation. These players will be managed with minimal additions to the main code (besides selecting the correct players - a single atomic operation from the GodOfWar (context) point of view, by rebinding one variable), and will not need large conditionals in order to make the appropriate decisions.

(Participants:

- Context: War
- State: Player
- ConcreteStates: User, CPU, GUIUser
- Client: TestingMain)

Unit Hierarchy State Pattern

Units can iterate through a state machine consisting of a civilian state, a deployed state with fit and injured sub-states and a dead state.
Changing the state of the object influences its run-time behaviour since units in different states deal different levels of damage and also influence the morale of a platoon and thus the country. By implementing a state pattern the need for various if and/or switch statements is eliminated when requiring units to do damage in attacks. The unit itself is the context in the state pattern and holds a pointer to its current state.

(Participants:

- Context: Unit
- State: UnitsState
- ConcreteStates: Fit, Injured, Dead, Civilian)

Country State Pattern

All countries enter the war as neutral and can then iterate through joined as well as withdrawn. This iteration is controlled by the country's morale and thus the country acts as context to its state and the iteration is automated. Only countries in the joined state can actively participate in the war, including recruiting new platoons, marching into areas, destroying transport routes and requesting factories. Countries withdraw from the war when their morale is too low, thereby disadvantages their alliance through the loss of areas and platoons.

The state pattern ensures that all of this management is automatically handled, without complex conditional logic, resulting in minimal changes to the context.

(Participants:
- Context: Country
- State: CountryState
- ConcreteStates: Neutral, Withdrawn, Joined)

Template Method Design Pattern:

The template method design pattern is used in the initialisation phase of the war simulator from the Player hierarchy:

The algorithm has pluggable steps for the creation of countries, alliances, platoons, initialisation of factories etc. The flow of the creation (ie. the order and procedure) is controlled by the Player class, while all the initialisation is handled by the subclasses. Therefore, the creation can be achieved by different types of players resulting in different results but following the same basic steps that are implemented by subclasses. The current algorithm consists of both an invariant operation (initialiseSide - which is in turn another template method) and variant operations, declared as pure virtual in the Player class.

This reduces code duplication as only a single instance of the common creation method and initialiseSide needs to be defined. Furthermore, it is easier to add or remove steps from the algorithm, since it can be achieved in one place instead of changing multiple subclasses.
The parts that are variant are separately maintained by the subclasses User and CPU, where User allows user input while CPU initialises automatically. This increases the maintainability of the code and eases debugging since it is easier to pinpoint where the mistakes are.

Furthermore, this also allows us to extend the setup capabilities so that different subclasses of players can set up the war simulator in different ways through a common interface, thereby only needing to reimplement the parts of the algorithm that vary. This can result in diverse war simulations with the addition of a single, simple subclass.

This also reduces coupling as the classes that call the template method need not explicitly know which Player it is calling the method on. They also need not know about all the substeps (which are protected), but only about the template method. This, combined with the state pattern mentioned above, ensures that all classes in the system only need to know about the Player (abstract) class.

(Participants:
- AbstractClass: Player
- ConcreteClass: GUIUser, User, CPU)

Visitor Design Pattern:

The visitor design pattern was used in our war simulator to iterate through map components including areas and transport routes. This was in order to keep our composite map stable while being able to add additional operations to its components. Specifically we used the visitor to determine how many areas did certain alliances control and to clean up dead platoons in an area and set attackers to defenders.

An important application that was implemented was to check who was the winner of the war by checking the spread of the alliance across the map.

For each additional feature for area and transport routes, a visitor was created in order to logically group these operations together so that the map could have one reference point so that the logic wasn’t scattered through areas and transport routes.

The major importance of the pattern was to allow our map to add functionality to the areas and transport routes without doing so in the respective classes. This decoupling ensured a stable system where if a certain visitor resulted in errors, we could isolate the visitor and fix the errors without being worried about the interaction of all components and/or destabilising the system.

(Participants:

- Element: MapComponent
- ConcreteElements: Area, TransportRoute
- Visitor: Visitor
- ConcreteVisitor: AreaVisitor, CountVisitor, CleanUpVisitor
- Client: Map)

Strategy Design Pattern:

The strategy design pattern was used in our platoons weapon strategy in which the platoon could choose to either deal damage using precision weapons or explosives.

The strategy pattern allowed for there to be an interface to alter between the two strategies without having to code if statements or switch statements - this decreases management complexity since toggling is implemented in subclasses and eliminates the need for parsing in order to check type. The reason as to why strategy was used, was because both weapons followed the same routines and interactions with the platoons. However the damage dealt and the ammo used was different for the strategies. Thus this is why we used the pattern, as the implementation we used followed the intent of the pattern.

The pattern also allowed for us to shield the client from the changes of the algorithm and the system would remain stable even when the derived classes would change their strategy.

We could also plugin more strategies without having to change client code therefore resulting in a scalable system.

(Participants:
- Context: Unit
- Strategy: PlatoonStrategy
- ConcreteStrategy: PewPewAttack, BoomBoomAttack)

Composite Design Pattern:

Our system encompassed the Composite design pattern in the Unit hierarchy where the class Platoon is the Composite, Unit is the Component, and Human and Vehicle are the leaves of the Composite tree structure. Our system followed the composite design patterns intent as the Platoon held compositions of the Human and Vehicle objects while both leaves were still allowed to be interacted with the client individually.

The advantages that this pattern brought to our system is that we can create a platoon with the vector of humans and vehicles which would then allow us to interact with the vectors as only one object. This allowed for the platoon to be scalable as the client could introduce a new leaf to the system in which the platoon could take it in as a vector. Furthermore, a client could interact with the composite platoon in the same way that it would with any other component in the Composite hierarchy. This eliminates the need for complicated if statements, querying pointers for their type and dynamic casting by providing a uniform interface for all operations that are needed in the hierarchy.

Another benefit is that the platoon would be dynamic in the classes it inherited from the Unit as the platoon would have to take the total health, morale, damage, etc from each vector of humans and vehicles. Thus if one human or vehicle is dead/destroyed then the platoon would change its health and damage.

This advantage allowed for the user interacting with only one needed interface and not needing to create functions in both leaf nodes and only in the platoon.

(Participants:
- Component: Unit
- Composite: Platoon
- Leaf: Vehicles
- Leaf: Humans)

Decorator Design Pattern:

In order to render composites from the unit hierarchy on the graphical user interface they need to have an appropriate texture set. Appropriate textures include the land platoon texture and the air platoon texture. We therefore make use of the decorator design pattern to set appropriate textures..

The “Branch” decorator along with the “LandBranch” and “AirBranch” concrete decorators separate the “nice to have” classification of units as well as the “nice to have” texture selection from the unit or composite itself.

(Participants:
- Component: Unit
- Decorator: Branch
- ConcreteDecorater: AirBranch, LandBranch
- ConcreteComponent: Platoon, Human, Vehicle)

Prototype Design Pattern:

In order to minimise time-consuming creation procedures, a prototype design pattern has been implemented to clone Factories when requested from adjacent areas.

Since this request happens frequently and different can be requested, but only if that factory exists in an adjacent area, the Prototype design pattern simplified the logic required to create the correct factory, by cloning factories already existing in the area from which it is being requested. There are only three factories and thus there is a small set of standard variations of the Factory object and thus the Prototype can automatically handle the logic as to which one to create, instead of having complicated switch or if statements.

This results in a more generic interaction with the factory hierarchy, that can simply call the clone function without explicitly needing to define the type of factory required and calling its constructor. The correct factory will then be cloned and returned, resulting in decreased complexity and increased flexibility. This increased flexibility is realised in the fact that more factories can be added, different constructor methods can be defined and the factories can even be swapped out without touching client code.

This is more manageable and results in a more modular system where the factory hierarchy can be managed independently from the rest of the system, easing debugging and making the system more scalable.

(Participants:

- Prototype: TransportFactory
- ConcretePrototype: GTFactory, PTFactory, ATFactory
- Client: Area)

Builder Design Pattern:

The builder design pattern allows us to easily and simply create required plattons. The builder was chosen for this task as we could easily separate the construction of the platoon from the representation of it. So we can just leave the builder to do all the logic and creation while the client, in this case country state, does not need to see or understand it, they just get a finished product.

The Builder design pattern gave us finer control over the creation process. As we build the platoon step by step. It allows us, if we want to expand our project, to easily expand and create extra steps without doing a major code refactor.

We implemented the builder pattern in our project to build platoons. We have multiple steps in the creation process, namely:

- setCountry,
- createSoldiers,
- createVehicles,
- createAmmo,
- createPlatoon,
- setBranch.

Each of these steps define a variable inside of the platoon, thus setting or creating each respective component required by a platoon.
This system is extremely modular, as we can simply add or remove steps without affecting the entire system, as it is encapsulated and separated from the rest of the project.

(Participants:

- Builder: BobTheBuilder
- ConcreteBuilder: PlatoonBuilder
- Director: Director
- Product: Unit
- Client: Country)

Singleton Design Pattern:

The GUI implementation of the system requires each object that is drawn on the screen to have a sprite and texture. Textures are what sprites use as their appearance when being drawn. Sometimes it is necessary to change the texture of a sprite. For example, when a country is occupied it must load the correct colour texture for its tiles. Depending on how it is done, texture loading can take significant time.

Textures can be loaded from files or updated from image objects. Loading textures from files requires a costly i/o access whilst updating textures from images objects only accesses RAM. However, image objects must also be loaded from file. This means that it is more efficient to update textures from image objects only if the image objects have already been instantiated.

This leads to the implementation of an image library class where images are stored in a hashmap to be loaded in the future. We use the singleton design pattern because a single library should be globally accessible to all sprites requiring texture. It would be inefficient for each drawable object to have a unique image library because the same image can be applicable to multiple drawable objects and therefore could be loaded in twice.

The use of a singleton design pattern is justified here because the image library should exist for the entire lifetime of the program’s execution and should be a unique but shared resource which stores image objects.

The result is streamlined access to a cache of images which greatly improves the fluidity of the graphical user interface over the previous i/o calls.

(Participants:

- Singleton: ImageLibrary
- Clients: Unit, Area, Coordinate)

Mediator Design Pattern:

As may have been mentioned by now, wars play out on large expanses of land called maps. These maps are made up of Areas which are connected in an adjacency matrix of TransportRoutes which are both of type MapComponent.

Areas and TransportRoutes are held inside the map and communicate with each other through notify-esq methods inside their shared map. For instance the destroyTransportRoute inside map method can be called from an area on two target areas. The map then notifies the target areas to destroy the transport route connecting them. Many other methods such as this exist within this mediator pattern.

This mediator pattern gives us the ability to change MapComponents in any way without necessarily requiring a change to the other MapComponents, instead changes only require updates to the map class.

Additionally, the decoupling of the MapComponents from one another increases their individual cohesiveness contributing to their reusability.

Moreover, using the mediator pattern allowed us to transform the many-to-many relationship that exists between the MapComponent, in which each area would have to reference each other area, to a one-to-many relationship, in which each area only has reference to the map that it is in. which is easier to understand and maintain.

(Participants:

- Mediator: map
- ConcreteMediator: map
- Colleague: MapComponent
- ConcreteColleagues: Area, TransportRoute)

Factory Method Design Pattern:

The factory method design pattern is used to create transport for people, ammunition and goods which is a crucial part in the war simulator. Ammunition is used for the weapons for the platoons to use against the enemy platoons, people (medics) are used to heal the platoons and goods are used to boost the morale of the platoon.

Instantiation is deferred to the subclasses, allowing the subclasses to decide the correct class to instantiate which removes the responsibility from the user of the factory. Factory method simplifies the creation process as no knowledge about the creation process is needed, only a single function call which returns the correctly constructed product. Each derived factory creates its own product in a predefined way specific to itself, meaning that an ammunition factory cannot produce a good product and vice versa.

One can easily add more factories and products without major refactoring of the code base.

(Participants:

- Creator: TransportFactory
- ConcreteCreator: GTFactory, WTFactroy, PTFactory
- Product: Transport
- ConcreteProduct: Goods, Weapons, People)

UML - Link: https://github.com/sloththedev/cos214-project/blob/main/COS214-Final%20UML.pdf