Bachelor’s thesis

Information Technology

2018

Antti Oksman

GAME DESIGN PROCESS - REMAKING A DOS GAME

– case Triplane: Furball

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2018 | 47

Antti Oksman

GAME DESIGN PROCESS - REMAKING A DOS GAME

- case Triplane: Furball

The aim of this thesis was to introduce the various phases of the game development cycle, starting from the planning and design phase and ending in the almost ready-to-be released product. This was achieved by reviewing the phases only briefly, inspecting the different work stages step-by-step, the overall design and development process and some of the occurring problems and their solutions. The work conducted for this thesis was not based on any specific written theory for game development and relies more on the authors own experience and knowledge about game design and development.

The work was carried out for MansikkaMarmeladi.io, a game company whose first commercial product the subject of this thesis was. The game is based on a game called Triplane Turmoil, originally released for DOS (disk operating system). The gameplay mechanics and visual expression was borrowed from the original game and was implemented with a modern look and feel. The new game is otherwise completely made from the ground up with new original assets, audio, code, and graphics.

The game was evaluated at different phases of development with user testing by the commissioning company and external testers. The company also attended several demo days and different game events where feedback about the game was collected. The game proved to be a very functional and entertaining concept.

In the time of this thesis work, the game was not completed fully to be available as a published product. It is difficult to estimate the amount of the remaining work, however setting the schedule for releasing the game in early 2019 should be possible.

KEYWORDS:

game design, game development, retro, GameMaker

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut järjestelmät

2018 | 47

Antti Oksman

DOS-PELIN UUDELLEEN SUUNNITTELU JA TOTEUTUS

- case Triplane: Furball

Tämän opinnäytetyön tavoitteena on esitellä pelikehitys- sekä suunnitteluprosessin eri vaiheet projektin aloituksesta toteutuksen viime metreille saakka. Vaiheet käydään pääpiirteittäin läpi, tarkastellen niiden eri työvaiheita sekä niissä esiintyneitä erityisiä ongelmatilanteita ja niihin ratkaisuja. Työ ei perustu mihinkään tiettyyn pelikehityksen teoriaan vaan pohjautuu enemmän kirjoittajan omaan kokemukseen sekä tietämykseen pelisuunnittelusta ja kehityksestä.

Työ tehtiin MansikkaMarmeladi.io -nimiselle pelialan yritykselle, jonka ensimmäinen kaupallinen tuote oli tämän opinnäytetyön aihe. Työn aiheena oleva peli perustuu peliin nimeltä Triplane Turmoil, joka julkaistiin alun perin DOS-käyttöjärjestelmälle. Pelimekaniikka sekä visuaalinen ilme lainattiin alkuperäiseltä peliltä ja toteutettiin modernilla tuntumalla sekä ulkoasulla käyttäen täysin uusia grafiikoita, koodia ja musiikkia.

Peliä arvioitiin sen eri vaiheissa käyttäjätestein, jossa testaajana olivat toimeksiantajayritys sekä ulkopuoliset testaajat. Yritys osallistui myös useisiin demopäiviin sekä erilaisiin pelitapahtumiin, joissa kerättiin palautetta pelistä. Peli osoittautui hyvin toimivaksi ja viihdyttäväksi tuotteeksi.

Opinnäytetyön puitteissa ei peliä saatu vielä julkaisukelpoiseksi tuotteeksi. Lopullisen jäljelle jääneen työn määrää on vaikea arvioida, mutta pelin julkaisemisen aikataulun asettaminen vuoden 2019 alkupuolella pitäisi olla mahdollista.

ASIASANAT:

pelisuunnittelu, pelikehitys, retro, GameMaker,

CONTENTS

FIGURES

IMAGES

GLOSSARY

Asset	Assets include everything from 3D models, sprites, sound effects, code and anything that could be used inside the game.
DOS	Disk Operating System, initially released in 1981
Freeware	Software that is distributed with no monetary cost, all modifications are prohibited without permission.
Github.com	A web-based hosting service for version control using Git. It is mostly used for computer code.
GPLv3	General Public License, software released under GPL usually includes all source-code and resources free-of-charge and can be modified freely
Indie game	A game that is published or produced with no outside funding.
Shareware	Proprietary software, which is initially free, and user are encouraged to share it. Usually not having all the features included/enabled.
Skirmish	A term used in older games to describe a mode where players battle against each other.
UI	User interface, space where user interacts with the game.
Game engine	a software development environment that can be used to build video games for different platforms.
GMS	GameMaker Studio, game engine developed by YoYo Games Ltd.
GML	GameMaker Language, programming language used especially in GMS.
JSON	JavaScript Object Notation, lightweight data-interchange format. Easy for humans to read and write, Easy for machines to parse and generate.
Memory leak	A failure in a program to release discarded memory, causing impaired performance or failure.

1 INTRODUCTION

This thesis aims to overview the whole game design process from start to finish. As the subject is very broad, each individual step will be briefly discussed, not delving too deep into specific subjects or details about the process.

The client for this project is an indie game company called MansikkaMarmeladi.io, which are developing a game called Triplane: Furball that is a remake of an old classic Finnish indie game called Triplane Turmoil. A remake of a game means that the game is heavily influenced by the original game but is a separate stand-alone product. The author of this thesis works as the lead designer for the company and is also responsible for coding of the game. Triplane: Furball takes place approximately in the WW1 era when triplanes dominated the skies. The game features 4-player skirmish battles against AI and/or human players and a difficult single-player campaign that should take at minimum 5 hours to complete. The gameplay purely consists of the dogfighting of these fighter planes and the name of the game comes from aviation slang where furball means "a confused aerial engagement with multiple combatants".

The development process used in this thesis is a product of accumulated knowledge from various sources like online videos, talks and from overall experience about games, more than depending on set rules or actual written theory. The thesis will be reviewing the steps taken in the design process and comparing these to the set guidelines and the common practices that are used by the game industry.

This thesis will examine some very specific and most interesting problems encountered during the development process and how these could be solved. Some of the problems are common in the sense that they could be encountered in any game development process and some only apply to this specific project.

2 DESIGN SPECIFICATIONS

When designing a remake, nostalgia can be one's best friend and worst enemy. The feeling of nostalgia can be deceitful for those who experience familiarity with the product as this can trigger emotions and feelings that accompany these feelings that could lead to the product not feeling as good as the original used to. It is essential to take the needs of modern audience into account and use them together with the feelings that raise familiarity.

The goal of this thesis is to design a game that gives the player that familiar feel of nostalgia and keep the game fresh and interesting for the modern player. A DOS game called Triplane Turmoil (Image 1) will be used as the basis for the overall design and keep the key gameplay elements that made Triplane Turmoil the game it is. A strong trend of re-releasing old games can be observed in today's game market. Crash Bandicoot Trilogy is one the most recent examples of this trend. The team behind this launch Vicarious Visions only created new high-definition assets and released the game on all the main platforms and became an instant success overnight, selling millions of copies. (forbes.com – Crash Bandicoot).

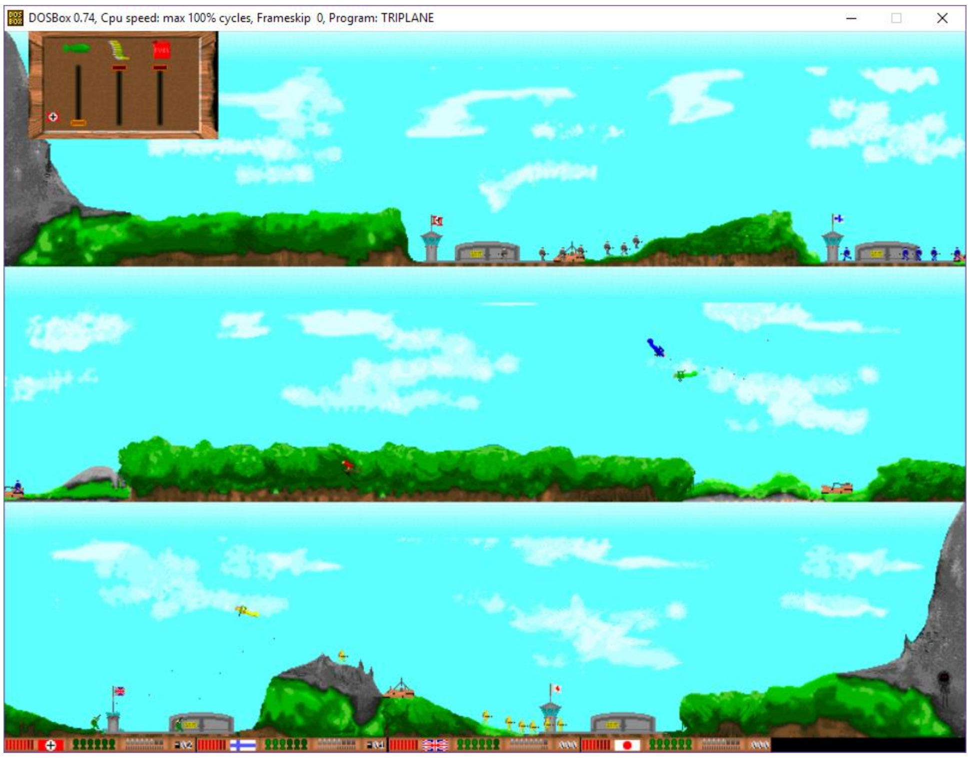

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TRIPLANE

Image 1. Triplane Turmoil

While this method of game design can be lucrative, work in this thesis aims for more originality in the product by taking a game that was made 20 years ago, modernizing the core concept and adding new features. At the end of the project, the game should have something familiar to those who know the original Triplane Turmoil and something fresh for those who are experiencing Triplane: Furball for the first time ever.

2.1 Triplane Turmoil

Triplane Turmoil was released in 1996 for DOS by Finnish game studio Dodekaedron Software. Triplane Turmoil featured a 4-player deathmatch against AI or PVP and a hefty single-player campaign mode. The game was originally released as shareware for the price of 100FIM equivalent to roughly 20€ today. Triplane Turmoil was later released as freeware and in 2009 Dodekaedron Software released Triplane Turmoil under GPLv3 license and it is currently hosted on github.com and is still somewhat actively updated. (github.com - Triplane Turmoil).

The events of Triplane Turmoil take place sometime during the WW1 era. The game features 4 feuding countries: Finland, Germany, Japan and the United Kingdom. The story mode consists of 4 different campaigns, one for each country with 6 missions each, totaling for 24 missions. The deathmatch mode is also known as skirmish, where maximum of 4-players can battle against each other or AI-controlled planes in 6 different maps. Each map features a base which is usually defended by anti-air cannons, machine gun turrets and patrolling infantry.

The most memorable element from the game was the cheery military marching music that was used as the main theme for the game and each country also had its own modulated version when playing the campaign. The music was composed by Markku Rankala aka ‘Dragst’ who worked for Dodekaedron Software. Everyone who has ever played the game can imagine the theme currently playing. The second unique feature was how the skirmish mode was laid out in three screens or planes in top of each other. When the plane would fly out from the top right edge of the map, it would re-appear from the left center edge etc. At the time this was extraordinary to see the whole map on the screen and how the players could make strategic decisions based on the information gathered from the available view.

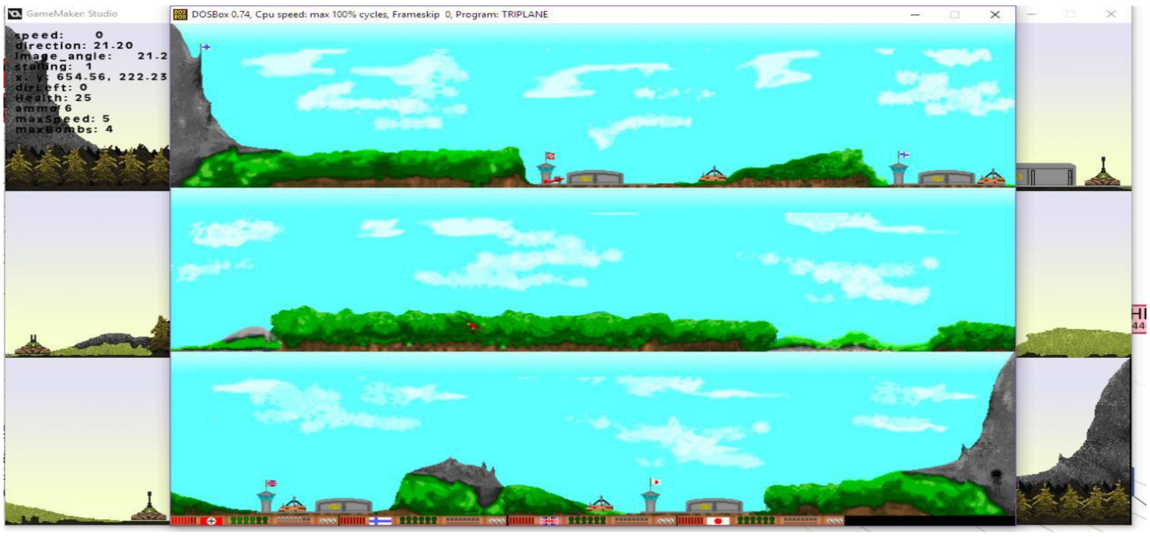
2.2 Modernization

The re-designed game should appeal to both larger target groups, the modern players that never played the original Triplane Turmoil and the nostalgic players who have previous experience. The design is focused on improving and adding on the following key elements:

a) Fast-paced combat
b) Eventful gameplay
c) Easy-to-use main menu
d) Intuitive in-game UI
e) Modern retro visuals
f) Unforgettable soundtrack

Fast-paced combat

Players today have accustomed themselves to fast feeling gameplay, compared to the old games which can feel sluggish and slow-paced. The resolution available in monitors today are a multitude of the DOS era monitors and with increased computational power, games have just become smoother and faster experience overall. Because of the increased resolution, game world can be made much larger and wider than the original had. (Image 2.)


GameMaker: Studio
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TRIPLANE

```
speed:     0
direction: 21.20
image_angle:    21.2
stalling:  1
x, y: 654.56, 222.23
dirLeft: 0
Health: 25
ammo: 6
maxSpeed: 5
maxBombs: 4
```

Image 2. Native resolution width difference

Creating larger sprites ensures that players who want to play the game from their huge 65" TV screens can also see where they are flying compared to the original DOS game which never had to be designed to be played in a living room. The original plane sprite is 20x20 pixels in size. A plane sprite of 36x36 pixels is used in this game to ensure the visibility of the sprite. The original game map was able to vertically fit about 7 planes whereas the new re-designed map can only fit about 4 planes (Image 3). To compensate for wider and lower map sizes, the planes need to have more speed added to keep the gameplay feeling fast and for the players to be quickly able to fly to the opposite ends of the map. The agility of the planes had to be increased so the players can control the plane that has greater speed. This results the gameplay to be very intense as the space is limited and decisions need to be made swiftly when players are in a combat situation.

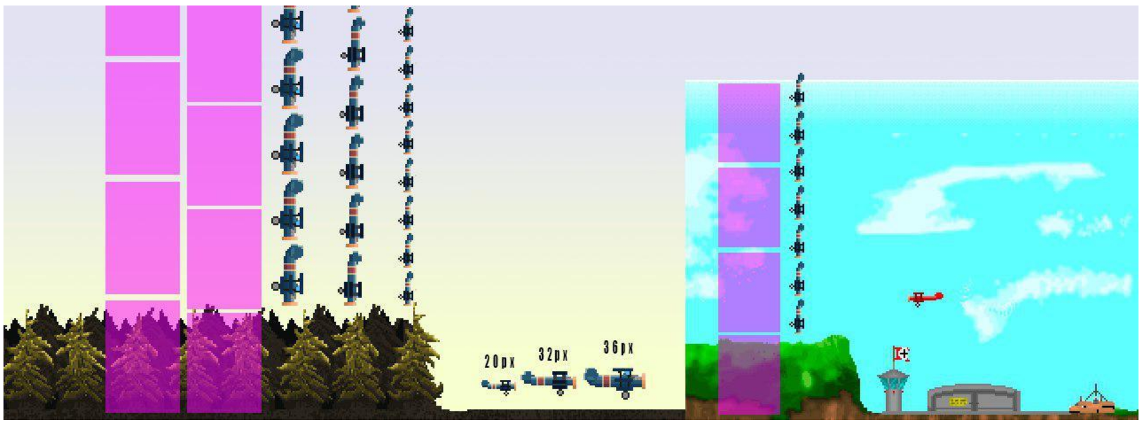
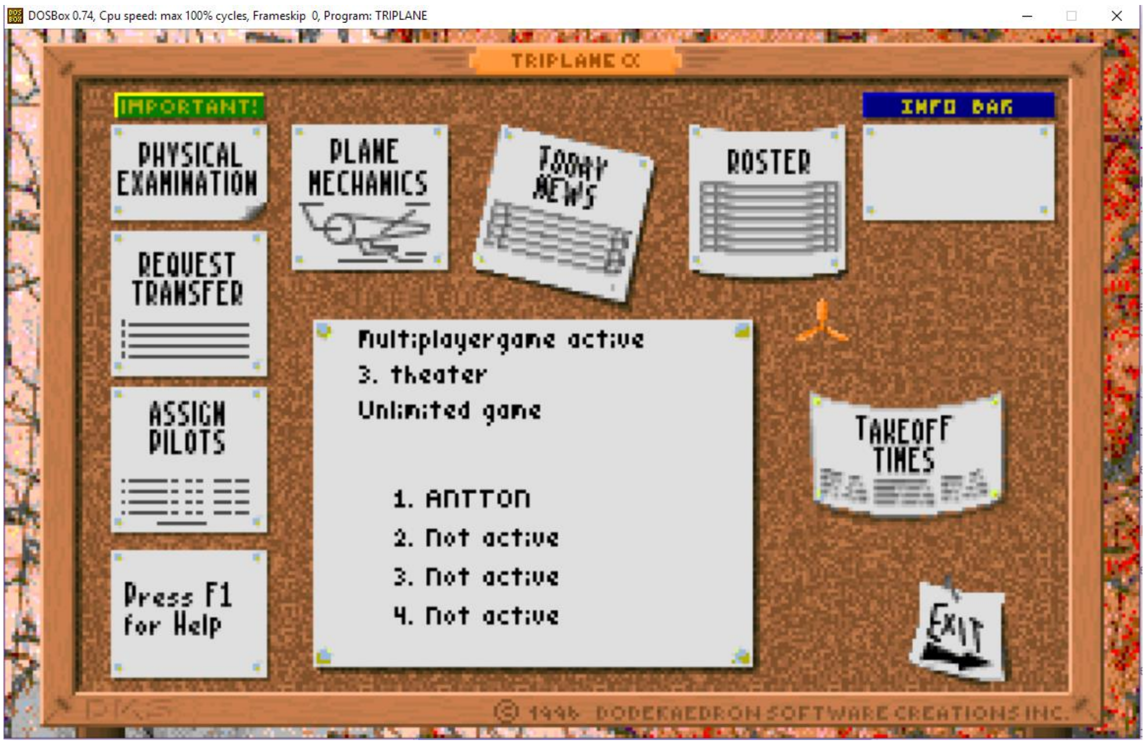


Image 3. Comparison of map height.

New UI

The original UI is very clunky and hard to use for those unfamiliar with it (Image 4). While it had its good side or maybe just a golden memory from the days of being young and adventurous, it was hard to navigate and would sometimes function illogically, for example when setting up the controls for the player, then selecting that player in skirmish and finding that the controls will not work. This was because each plane controls had to be set separately in a controller menu specific to skirmish and the controls set for individual player would only work in solo mode play.



Multiplayergame active
3. theater
Unlimited game

1. ANTTON
2. Not active
3. Not active
4. Not active

Image 4. Triplane Turmoil UI

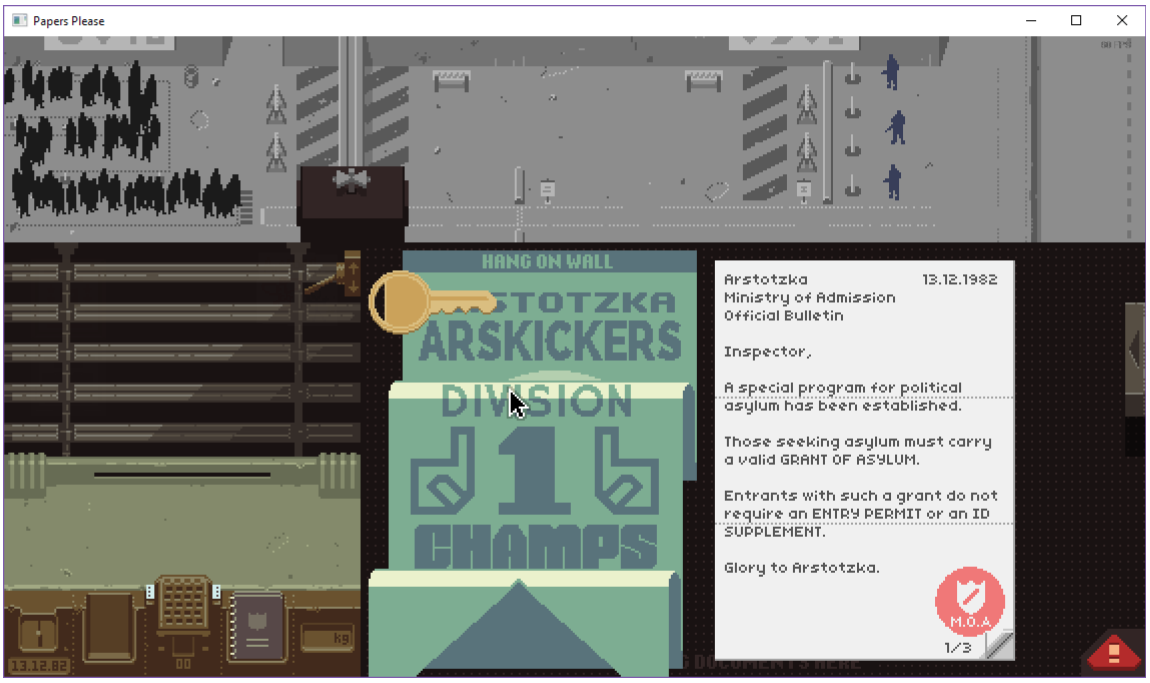
The old UI can be used as a good reference image for the style of the new design. Because modern hardware does not have the limitations of color or resolution, designing a good-looking UI is much easier. A modern UI should use elements which the players are already familiar with, to make it intuitive and easy-to-use, for example, the concept of buttons and how they function or if a box looks like it can be toggled, then it should also function that way.

When designing the menu, it is important to take in to account the living room element and to ensure a good user experience when the player is using a gamepad only. Buttons need to be drawn large enough to be read from two meters away and the whole menu (and the game) must work without the need to use a mouse for navigation. Menu paths should be simple, and elements must have clear visual feedback when they are selected and when then they are not. A well-designed UI makes the game experience feel polished and players not feeling frustrated.

Graphical style

The art style should remind of the original game, while bringing modern elements and effects to the table and modernizing the overall visual style. This can be achieved, for example, by adding particle effects to visualize the game and therefore lessening the need for hand-drawing animations and effects used.

The graphical style will be drawing from the WW1 -era and games that take place during that era, keeping away from pastel-toned colors and keeping the style more dark and serious. The Medal of Honor series and Papers, please! are great examples of the style this game aims to emulate (Image 5).



Arstotzka 13.12.1982
Ministry of Admission
Official Bulletin

Inspector,

A special program for political asylum has been established.

Those seeking asylum must carry a valid GRANT OF ASYLUM.

Entrants with such a grant do not require an ENTRY PERMIT or an ID SUPPLEMENT.

Glory to Arstotzka.

Image 5. Papers, please!

2.3 New features

The intention is to make the game look like itself and not just something that was remade by copying all the assets from the original and writing some code. Every asset for the game will be made from scratch to fit the need and visual style, only keeping the look and feel of the original game. The original game featured 6 skirmish maps and one of these maps is re-drawn and featured as homage to the original game, also adding 5 totally new skirmish maps and a completely new single-player campaign, featuring 4 different personal stories and the story progression is designed for the players to feel the weight of the war.

The skirmish mode will have added mutators which can be unlocked by playing the story. Mutators are options that can be toggled to alter the PVP-gameplay in a multiple of different ways; for instance, removing the gravity all together, switching all plane bullets to bombs, making all machine gun turrets artillery cannons, having an AI wingman to help the players in the fight, kill streak bonuses and similar features.

During the ongoing development of a game, there are moments that produce ideas and improvements that could make the game play better, look better or add impressive features. This is called a ‘feature creep’ that will prolong the ongoing development when the team is deviating from the original plan. Making a solid design plan and following it from the start will help the game to become released and all those great ideas that come up during development can be saved for upcoming titles. As such, the original Triplane Turmoil was a solid game back in its days even with its flaws, so taking the set changes and keeping the core mechanics will avoid the feature creep while keeping originality for this new game.

3 SOFTWARE ARCHITECTURE

Creating games today is easier than ever. One does not really need any special skills or traits to do so. Just download the selected game engine and start hacking the game together. Internet is full of free assets, everything from art, sounds to complete games with source code included. Engines like Unity make it so easy that one does not even need to write code, just using a mouse to drag some 'blocks' around, glue some graphic assets on top and you have ready-to-ship game. Well in theory at least.

It is true that the barrier for entry is lower than ever and almost all the mainstream engines having large and helpful communities beside them; new aspiring developers create more games than ever before. This creates another problem of market saturation, while it is easy to start developing a game; it will be considerably harder to get it finished and even harder to get it seen by the potential customers. This is the part where one's maturity as a developer plays a big part and how the game is going to turn out.

3.1 GameMaker Studio 2

From the sea of engines, the one that stand out the most was GameMaker Studio 2 or 'GMS' in short. Developed by YoYo Games Ltd. initially released in 1999 and it has a superior track record with 2D games. (gamesindustry.biz - YoYo Games). GMS is also very affordable for small indie developers and with a free trial version that is more than enough to get one started in game development.

The main selling point for it was its usability and easy-to-learn interface. GameMaker being out there for so long it has gathered a big and active community with healthy amount of tutorial videos varying from beginner to advanced topics.

Comparing GMS to other similar engines like Godot, pygame and Construct and other 2D engines, (websitetooltester.com - comparing 2D engines) GMS felt like the most mature one and one with the ability to expand infinitely when knowledge of the engine increases. GMS can be used to compile your game to pure C++ and supports all the major platforms PC, MAC, Linux, HTML5, Playstation, XBOX, iOS and Android with Nintendo Switch support coming later this year.

3.2 GameMaker Language + JSON

However, biggest thing about choosing GMS over other engines was the programming language it uses. GML or GameMaker Language is derivate from C++, heavily modified one but as C++ syntax and programming paradigms were already familiar, the choice was made easy.

GML is friendlier when it comes to typing the language; there is no need to declare variables, memory allocation is much simpler, almost to a fault as GML does offer limited capabilities when memory management is in question.

There are some added constructs such as the 'with' block. (GML Manual - 'with') 'With' can be used to directly address 'objects' and change values in them. In C++ this would be generally be done by referencing the instance of that class and changing the value where ever it was referenced. GML allows lots of these types of shortcuts to make things little easier at the cost of complexity.

The ease of writing and non-typed architecture makes GML at times feel like a scripted language i.e. Python. GML also has built in data structures that resemble arrays but instead of allocating a certain amount of memory at run time are dynamic in nature. (GML Manual - Data structures) They are generally faster to manipulate with the cost of more memory usage. Data structures in GML can be also be imported and exported from and to JSON with building functions. This can be very helpful when the need of external files like a save game or hi-score lists are needed.

JSON itself is language-independent interchangeable data format for transmitting text from one place to another. With common usage and universal support for almost anything it is superior language to transmit human readable data over software application limits.

3.3 Development plan

Creating milestones is important part for the sense of advancement during development. While keeping all involved motivated when achieving these milestones, they're also important for scheduling the project. (Chandler 2013, 7.)

Rather than building discrete components and adding them together in the end, focus will be developing components than can be built on and added together during the development process creating a complete product.

First starting from a quick prototype with the plane and the basic mechanics like flying, turning, rolling the plane, then landing on a pad, adding the hangar where to 'roll in' and creating the hangar man to push player in and out. Bit by bit when building these components and adding on; product can be called a game or first playable prototype at least.

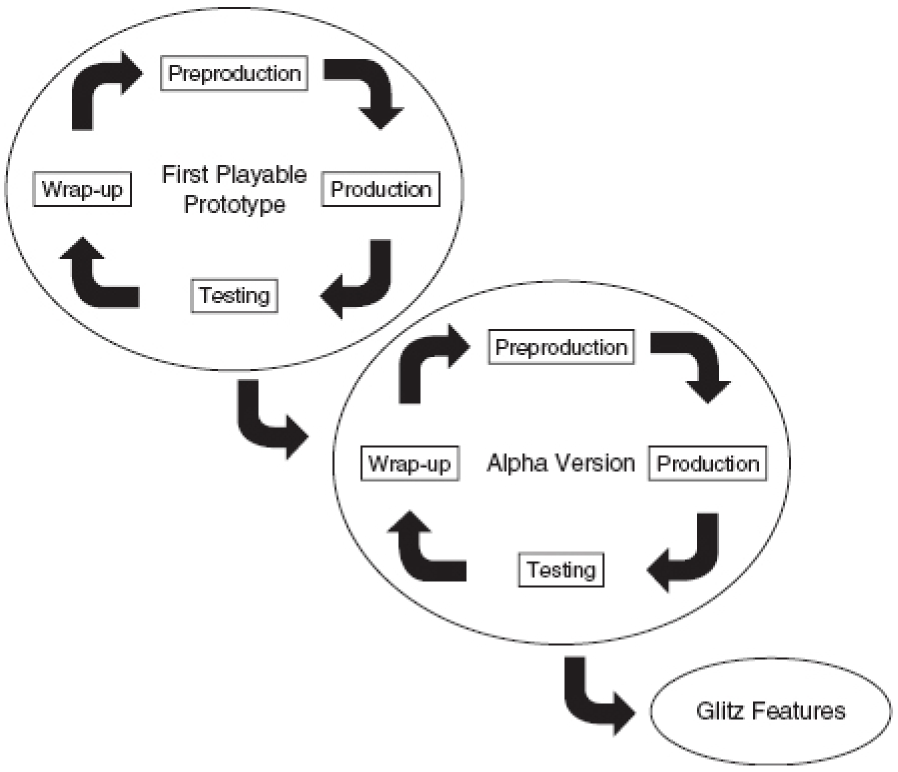


Figure 1. Multiple production cycles in a single product (Chandler 2014, 4)

After the basics have been implemented, project can move to larger complexities like building the maps, creating the menus, multiplayer, enemy planes and the rest of needed components. Iterating the same basic principle that can be seen in in Figure 1 for each individual component, each component can also be set as a milestone. Keeping the components small enough or chopping them to smaller milestones will make the workload also feel smaller and make the work easier to deal with in example doing the whole menu, its layouts and functions in one go. As the task is enormous, it would be very hard to work on and keep motivated. Instead chopping it in smaller individual task and creating milestones like in the example below keeps work much more manageable.

1. Menu design
2. Menu layout with placeholders
3. Menu functions
4. Individual menu location as separate milestones
5. Bigger menu items can be chopper to sub-milestones
   5.1. Options Menu layout
   5.2. Options menu function
   5.3. Options/Audio menu layout
   5.4. Options/Audio menu function

Manageability is a key, and this creates a natural way to break the work load and switch between different tasks. As production moves on, it becomes much easier to estimate the time it takes to complete something and to create these different milestones. Change of scenery also works in development, doing small task and switching between larger complexities keeps the work interesting through long development cycles.

Planning everything step-by-step and creating a path that must be walked exactly during development is either smart or good for developing a game.

4 IMPLEMENTATION

4.1 First Flight

The work started by borrowing the plane model from the original Triplane Turmoil, importing the sprite to GMS and creating base for the whole game. At this point there are no plans to future proof the code or making any elegant solutions, idea is just to start creating something that can be built on later.

Making the plane move on screen when arrow keys are pressed is simple enough, adding gravity, making the background blue to imitate sky and in under an hour the first working prototype is ready.

Adding a landing strip with a placeholder hangar, little collision system so the plane can land on the strip and 'Butters' the hangar man to just stand there, the landing strip is looking crowded already (Image 6).

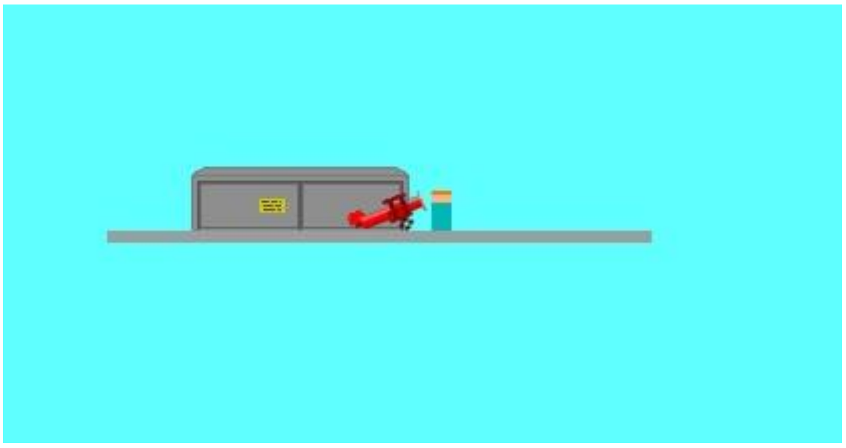


Image 6. Prototype ready.

4.2 Physics

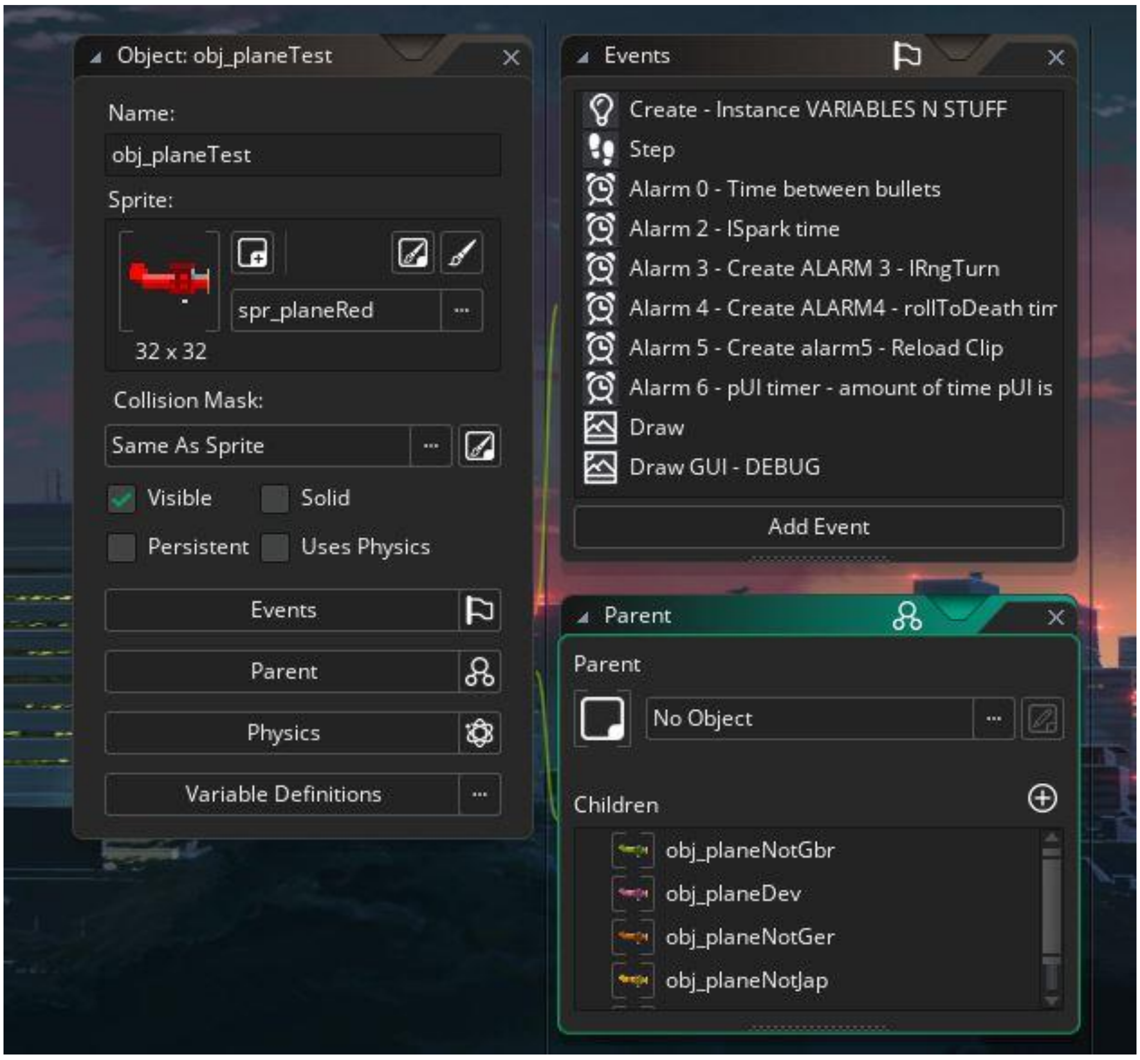
GMS has a readymade physics engine for beginning developers and quick prototyping but to gain as much control over the engine as possible, creating a dedicated physics engine in GMS to handle the needs of the game without the game engine itself limiting development.

In games real physics are rarely used and this was the case in this project also. Flying planes had to 'feel good' but not be limited by real-world rules. More than that not every object needs to have physics attached, so creating the engine was more done in a case-by-case basis and not as a global force affecting everything.

Planes

Planes need to be able to act like planes do; stalling, accelerate over max speed when flying down with throttle, glide without throttle, gain speed when gliding downwards, able to lose speed when gaining altitude, spin, roll, turn and all the other aerial maneuvers. The key mechanic in the game is that all the planes are different; they each have different weaknesses and strengths, so the flow and the dynamics are based around finding what strategy best suits the current situation in battle. All this needs to be considered when creating the physics for the plane, to avoid writing boilerplate code and to make fiddling plane stats easier when changes are wanted.

Creating a parent object that handles all those things mentioned above and more so that when a child object is created it will inherit all those properties from the parent. Making unique children this way is not that more difficult as the child will inherit the base stats and then with some set flags customize how the child is created. For example physics code is shared with every instance or object of type 'plane' and all different plane stats are in the parent object, when creating the child plane that is a different type of ‘plane’ it inherits the physics and the stats for that type of ‘plane’ (Image 7).



Object: obj_planeTest

Name: obj_planeTest

Sprite: spr_planeRed

32 x 32

Collision Mask: Same As Sprite

[✓ Visible] Solid

Persistent Uses Physics

Events

Create - Instance VARIABLES N STUFF
Step
Alarm 0 - Time between bullets
Alarm 2 - lSpark time
Alarm 3 - Create ALARM 3 - lRngTurn
Alarm 4 - Create ALARM4 - rollToDeath tir
Alarm 5 - Create alarm5 - Reload Clip
Alarm 6 - pUI timer - amount of time pUI is
Draw
Draw GUI - DEBUG

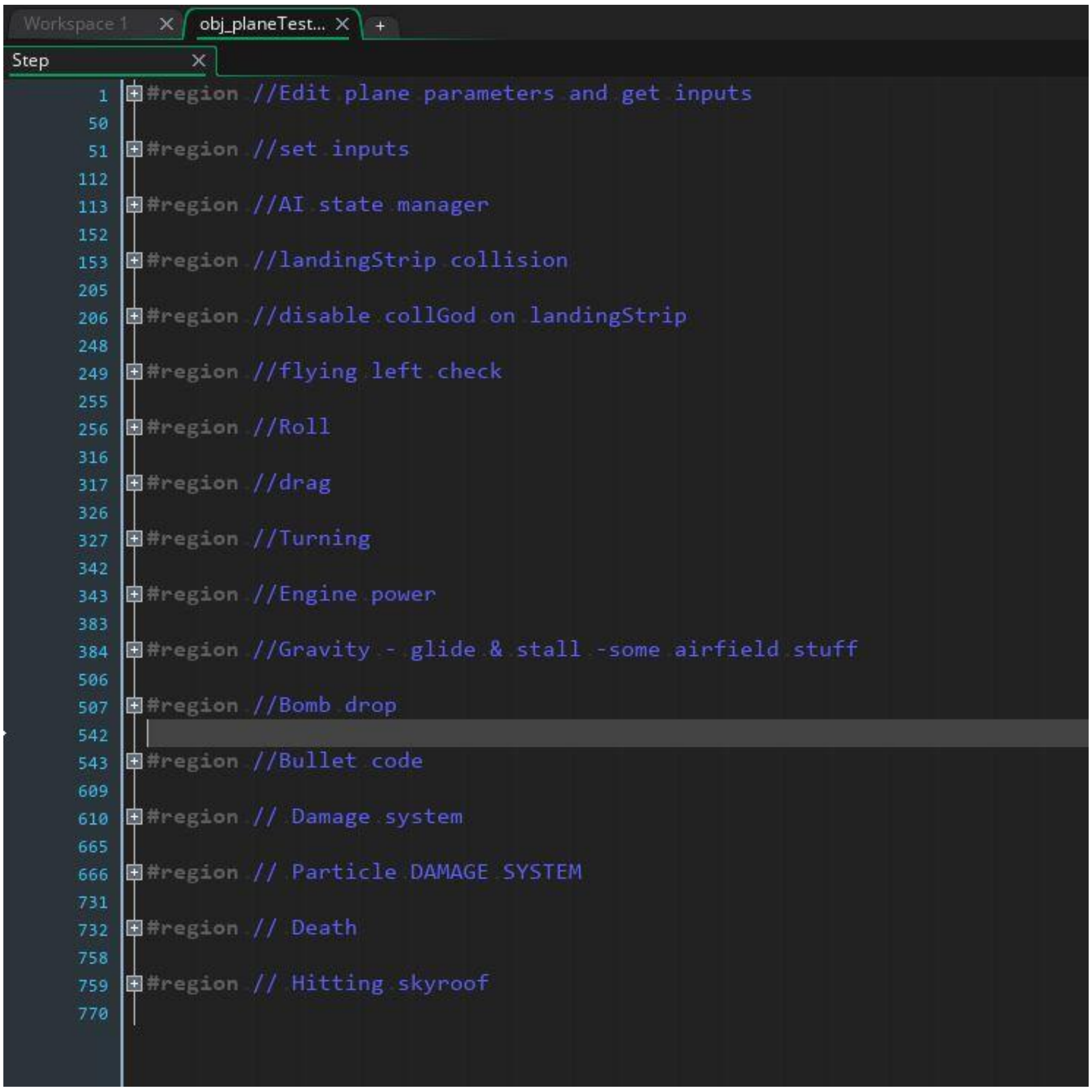
Parent

Parent: No Object

Children
obj_planeNotGbr
obj_planeDev
obj_planeNotGer
obj_planeNotJap

Image 7. Parent object of plane.

Writing a good collision system is the most important part of the physics engine, as all the planes need to react somehow with the world and objects around them. So, all the collision code that the plane object itself needs is written in the parent object, like everything else (Image 8).



```
#region //Edit plane parameters and get inputs

#region //set inputs

#region //AI state manager

#region //landingStrip collision

#region //disable collGod on landingStrip

#region //flying left check

#region //Roll

#region //drag

#region //Turning

#region //Engine power

#region //Gravity - glide & stall -some airfield stuff

#region //Bomb drop

#region //Bullet code

#region // Damage system

#region // Particle DAMAGE SYSTEM

#region // Death

#region // Hitting skyroof

```

Image 8. Step event of plane parent object.

What happens when plane touches another plane? When the plane hits the ground? How about flying too high or over the sides of the map; landing, gets hit by a bullet or a bomb? Everything needs to have a collision event, in principle it works by checking every frame if something has happened. GMS has a very good collision system built-in, so it is just a matter of design and coding how everything works and is handled. Famous scientist once said, "for every action, there is an equal and opposite reaction", in this case the latter is mostly true as this game does not need to obey the laws of physics.

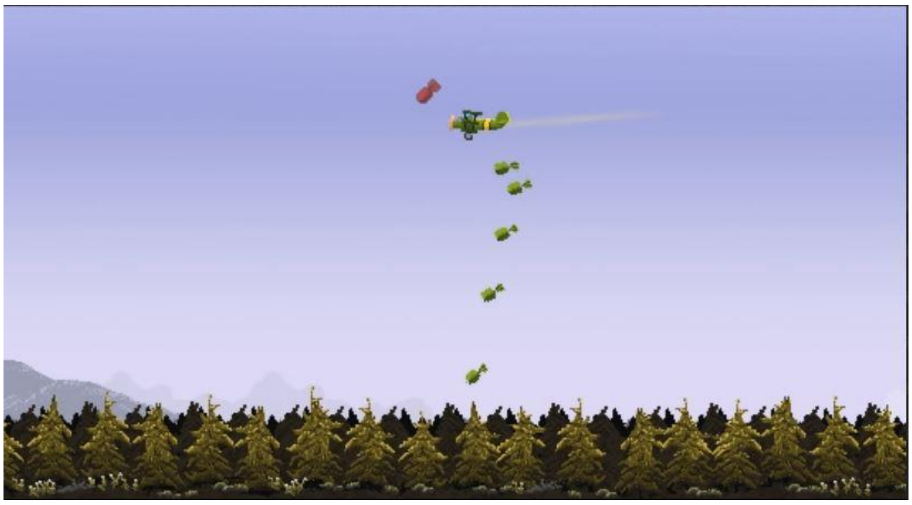


Image 9. Bombing the forest.

Bomb

Bombs are the other piece of the puzzle being the second mechanic that makes the game fun (Image 9). Getting the physics right was a priority so you would always feel a little chuckle when dropping one. Carrying more bombs adds weight to the plane making it heavier and cumbersome to fly. Players are needed to balance between agility and ability to suit their playing styles.

When designing bomb physics the goal was maximum fun, so there is very little drag added which results that they can be thrown almost half of the map length if done right. This creates moments of chaos and confusion as random player thrown artillery barrage of bombs will hit the base of opposite player (Image 10).

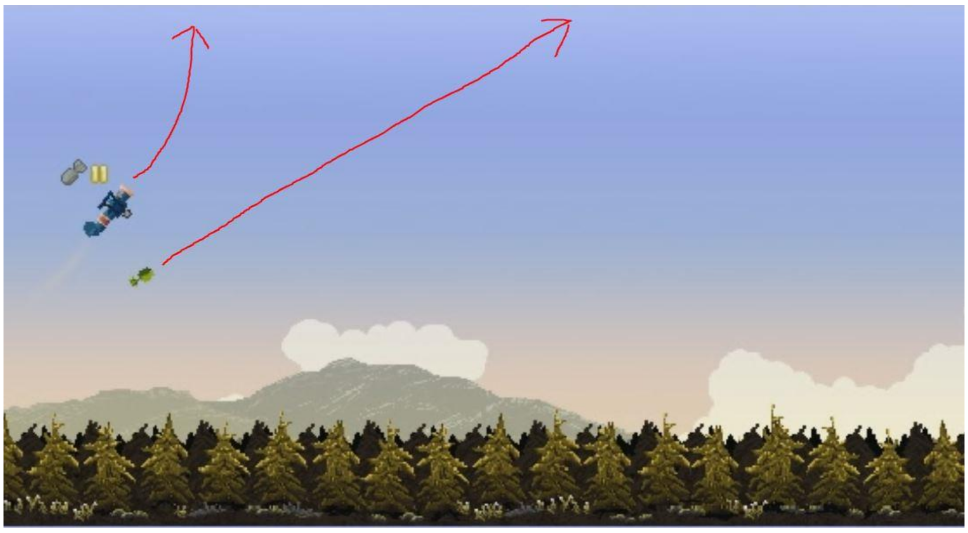


Image 10. Bomb trajectory.

Bombs of course also need their own collision events and as they destroy almost everything they touch and not just planes, similar decision was made with bomb collision as with plane collisions. Only thing that directly affect bombs like destroying them in mid-air would be handled in bomb code and everything else that is affected by bombs are handled in their own parent objects. For example, destroying another plane with a bomb, collision event would be handled in the parent event of the plane not the bombs. This eliminates the need for writing the boilerplate again by not handling every kind of parent object in the actual bomb code itself.

4.3 Multiplayer

With the basic mechanics implemented multiplayer component is ready to be built. Total of 4 players (human or AI controlled) can play against each other or fight in teams of 2 or 3. Each player controls 1 of the 4 sides available conveniently named after the color it represents; Orange, Blue, Green and Yellow. In the original game Triplane Turmoil each side was named after a country represented by its main colors and flag. A choice was made not to identify these different sides by name or a country or come up with any made-up country names or their back stories. However unique emblems for each side was created, which are more like a coat-of-arms than a flag for a country. This will help players to identify with the set aside more than just a color would (Image 11).

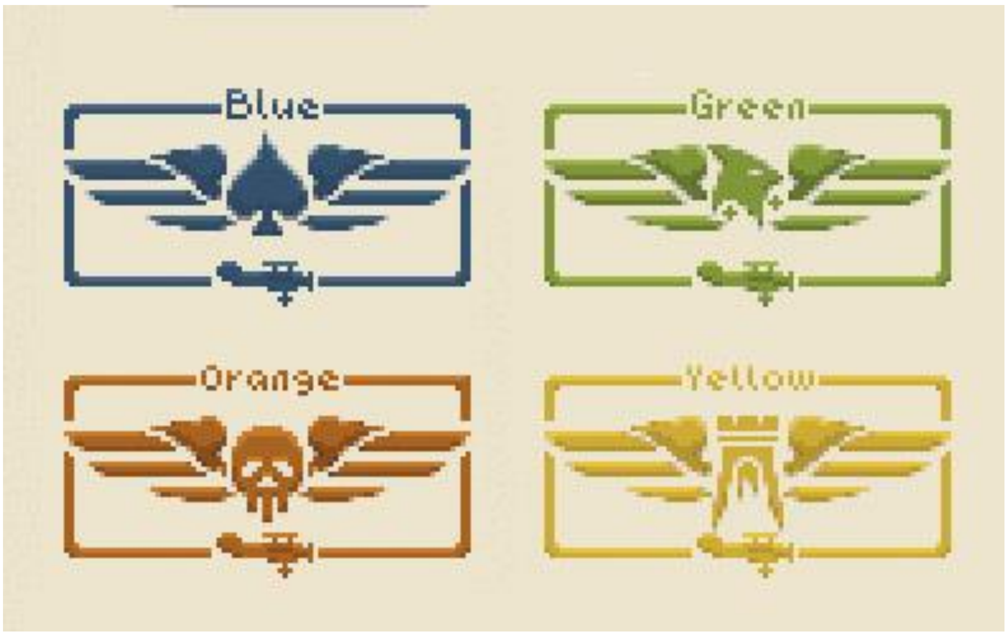


Image 11. Emblems.

After all players have picked their sides, they can choose the theater for battle. There are 6 unique theaters to choose from that each have their own unique elements, location and theme to make the battles play out differently and vary the player experience. Placements of the bases and their air defenses make the most difference. Some theaters feature unique elements (Image 12) like land forces battling each other and taking over the enemy base or in the desert theater there is huge rock formations where a player can hide behind and make a tactical strike to the unsuspecting victim or hirs base.

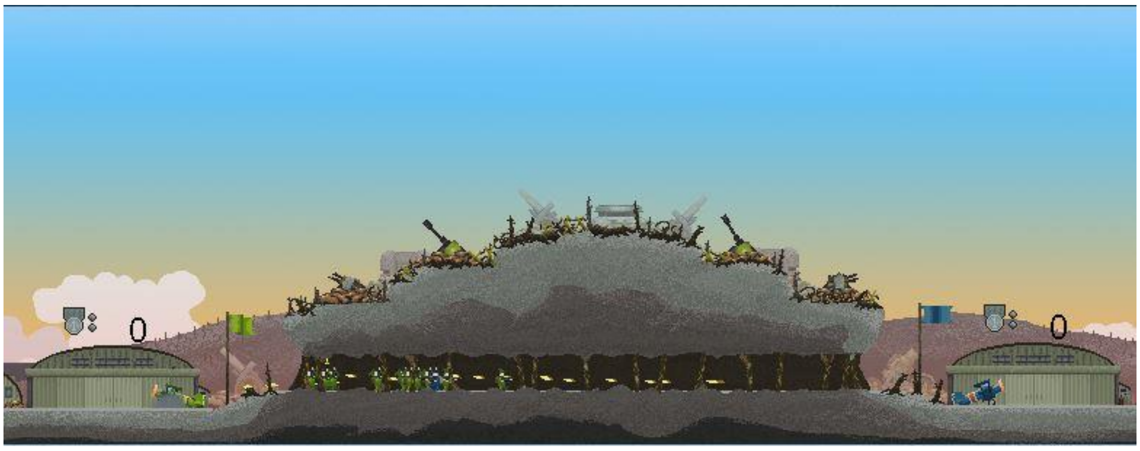
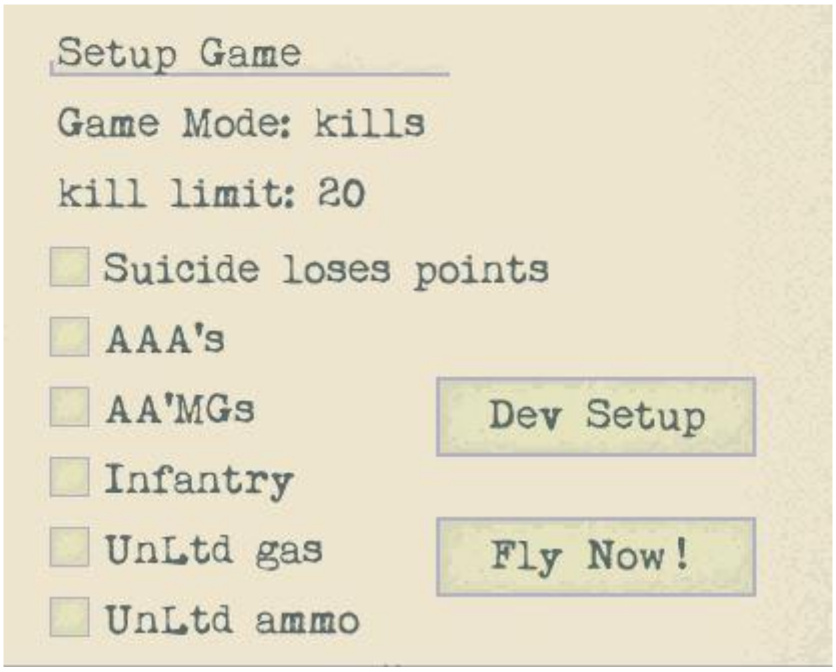


Image 12. Fly-through cave under a hill of anti-air guns.

Multiplayer features three different modes to choose from; kills, lives and scores. When playing for kills, a point limit is set and the player who first reaches the set limit by killing other players wins the game. In lives mode point limit sets the amount of lives each player has and when the lives run out hir is out of the game. Last man standing wins. Playing for scores players rack up points by bombing enemy base, killing enemy troopers and destroying enemy planes, first one to reach the set point limit wins the game. Fourth option is an unlimited game that only the game host can end manually.

Players also have lots of modifiers to choose from to make games more interesting. The basic ones include but not limited to; unlimited ammo, unlimited gas, disable infantry, disable AAA and similar modifiers (Image 13). There is also included some special modifiers like making all machine gun ammo from planes change to bombs, adding a fog of war so no one can see what’s happening on ground level, disable gravity, disable gravity for bombs only, troopers shoot mortars instead of bullets and similar modifiers. These special modifiers can only be unlocked by playing the campaign mode and are awarded when players reach certain checkpoints in the campaign or doing special tasks like flying from one end of the map to another with only one unit of petrol. Playing the game rewards, the player and breaks the normal gameplay-loop while also adding more gameplay elements that should add some longevity to the overall life of the game.


Setup Game

Game Mode: kills

kill limit: 20

☐ Suicide loses points

☐ AAA's

☐ AA'MGs

☐ Infantry

☐ UnLtd gas

☐ UnLtd ammo

Image 13. Basic skirmish modifiers.

4.4 Player base mechanics

Each player starts the battle from their respective bases. In addition to airfield and hangar, base usually contains at least two anti-aircraft weapons, machine gun turret, artillery gun, barracks that can spawn infantry and a control tower. Control tower is the only one that does not have any additional functionality other than providing points when destroyed in score limit game in skirmish mode.

Barracks

Barracks spawns a limited number of infantry to help in base defenses and in some maps and campaign missions to take over enemy bases and AA-weapons. When a trooper walks over enemy controlled AA-weapon it captures it to fight in their side until destroyed. This creates the need for player in certain situations to defend against enemy infantry. After a trooper is killed it spawns after some time has passed. Destroying the barracks stops the spawning of infantry and barracks does not re-spawn during the game.

Anti-air

Anti-air weapons are players' worst enemy and best friend, designing them not to be too accurate so that players would not feel them to be too unfair. Still for inexperienced pilots AA can prove to be a challenge. When in a furball over enemy territory, dodging the AA fire can become a real challenge even for the experienced pilot, so the fight is better take elsewhere. AA-MG can fire almost 180 degrees and have no height limitations on how low they start shooting. AAA has a bit lower range of motion but can shoot far greater distances and can't shoot targets that fly below certain limit (Image 14). They start tracking the target early, so it should give up player a little heads up that they are soon going to be under fire. However, it is usually too late if player passes that point and is not ready for it. AA-weapons re-spawn after some time when destroyed; this keeps the battlefield active and always full of danger. Re-spawning can be toggled off in the modifier menu.

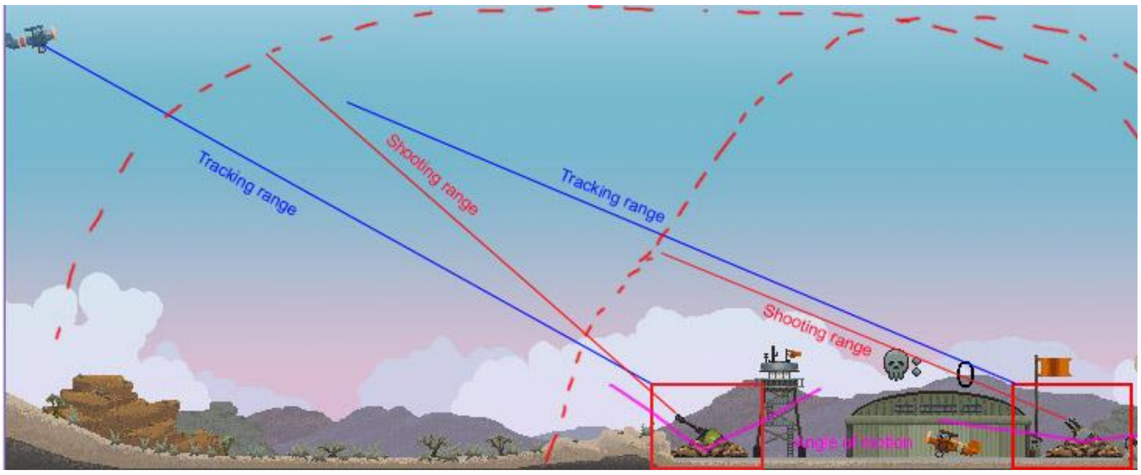


Image 14. Cannon ranges; AAA left, AA-MG right.

Hangar

Hangar is the place where player spawns from, after death there is a little timer that delays the time when player can spawn back in to the game. Amount of carried bombs, ammo and fuel can be changed in the hangar menu before spawning in to the battlefield. Your helpful mechanic will push you on to the tarmac and you can take off. If player takes damage in battle and manages to land in his own base, player can call the mechanic to push the plane back in for safety, repairs and reloading ammunition and fueling up the plane (Image 15).

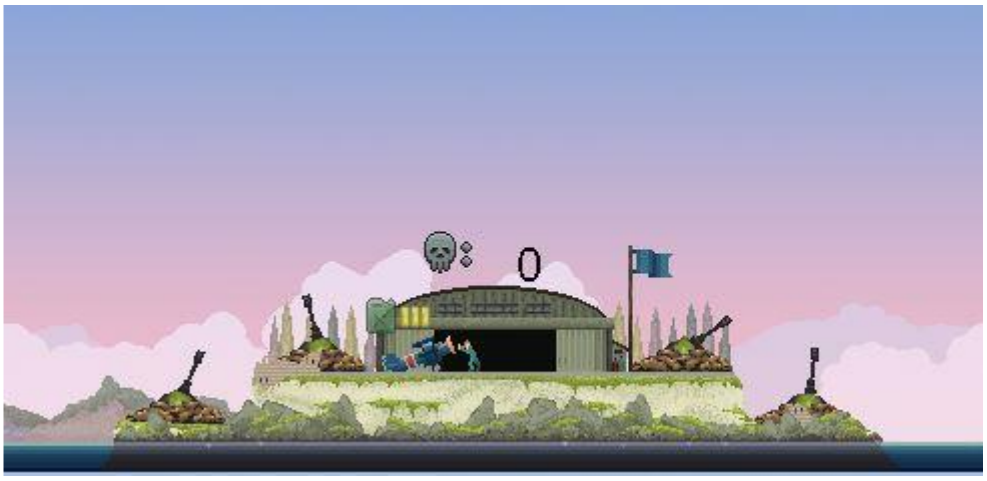


Image 15. Hangar man "Butters" in action.

4.5 User-Interface or ‘UI’

Players need to able to navigate through the game as much ease as possible, both in-game and off-game. Off-game meaning the interface that player is first greeted when starting the game aka 'Main Menu' and in-game interface meaning the things player needs to interact while playing the actual game aka 'HUD' or Heads-Up Display.

Main Menu was designed to look like a binder notebook; this had the unpleasant effect to unnecessary limit the space there was to use for menu items which made it somewhat crowded in locations in the menu. The goal was to keep menu paths as short as possible to reduce the need to go back and forth between things and for player to be able to get to in-game mode as quickly as possible. Before the addition of quickplay-button (1-click from menu), through skirmish menu it takes 2-clicks to get to in-game without making any changes to game setup (Figure 2).

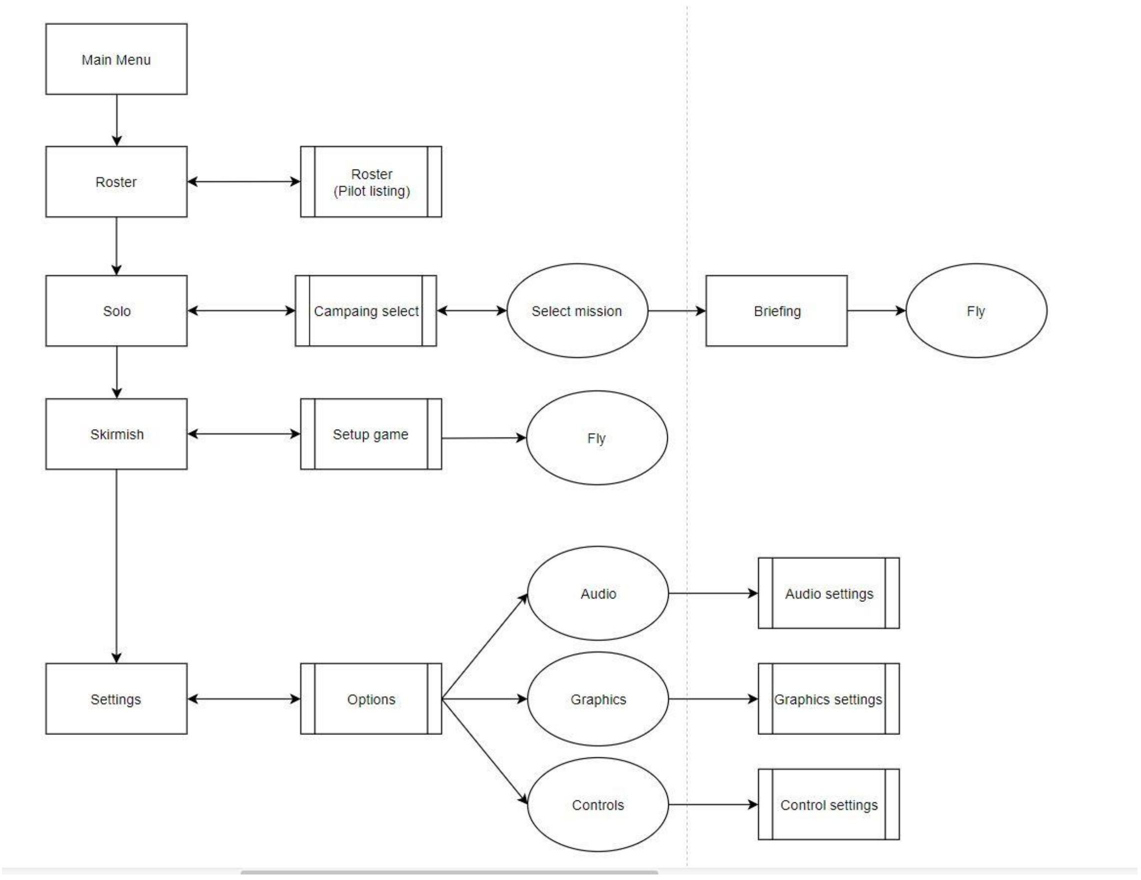


Figure 2. Basic UML chart for menu paths.

Quickplay button is a great way to introduce new players to the game. It is located conveniently on the upper left middle in the menu and the red color makes it catch your eye. It takes the player instantly to a balanced game with AI to get a quick glimpse of game mechanics, how things work and if they like the game or not. The first five minutes with a new player to the game are the most important ones and UI experience takes a big chunk from that short time span. (gamasutra, extra credits).

Everything is also made navigable with a controller only to support "TV-only", forcing the player to have a mouse for navigation would make the experience not-so-enjoyable. Having full controller support throughout the game also makes porting to consoles much easier job if that comes in question later in the games life cycle.

In-game interface (hUi) or the 'hangar User-interface' is where player can setup the number of bombs, ammunition and petrol for the plane before taking off. HUi consist of 3 vertical bars and each bar represents one unit of a type. Same icons are used in the player User-interface or ‘pUi’ to show the status of units and are only visible when consuming a type of thing like dropping a bomb or using enough petrol (Image 16). Making hUi floating above hangar and pUi following the player and being visible only short periods of time, this gives more space to fill the in-game room with more visually appealing content and the makes the actual interfaces less-intrusive.

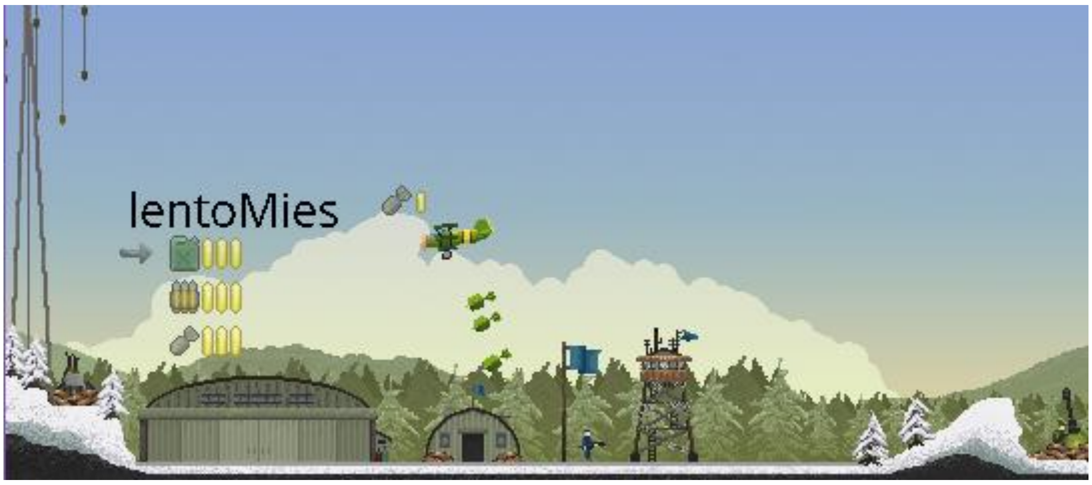


Image 16. hUI and pUI

4.6 Miscellaneous gameplay mechanics

AI

AI was designated to be a worthy opponent for players and to have different behavioral traits to accomplice different goals. This lessens to need to script AI planes in solo missions where the normal "seek & destroy enemy planes" is secondary and something else like "protect object x" is the primary objective. The base for the games AI was done by Sallamari Rantanen as a part of her thesis. (Rantanen, 2017)

Particles

Particles are created using the built-in particle engine in GML but to optimize the draw pipeline, memory usage and visual effects; custom particle sprites were created. This gives a lot more control about the look of effect. This enables the creation of all kinds of explosions, debris, smoke and weather effects that are more suited to the needs and visual style of this game.

Dynamic backgrounds

Dynamic backgrounds are designed to make the game world unique for every map. In addition to the random weather effects mentioned earlier, the background system can spawn different elements each time a session is started. This includes a random amount of clouds moving at different speeds, randomized color for the sky that gives the feeling of different times in a day, background elements like mountains or villages depending which map the player is currently playing and similar effects.

Audio

Sound and music generate the overall feel and the atmosphere of the game. The main track is the one player hears when starting the game. This cheery military march style track should give the feeling of lighthearted humor and fun. Another main track is the one player hears when starting the solo campaign. The campaign is set to be 24 missions' long starts with this uplifting 'ready-for-war' styled hi-tempo marching music but when the player makes progress through the campaign the same track is modulated and changed to sound a lot gloomier and depressing. In the end when player reaches the final stages of the long campaign the weight of the war should really be heard from the music alone.

There is no in-game music track as all the sounds of reloading, gun fire, bombing and the occasional scream of terror really set the feel for theatre of war.

Environment

Destructible environment makes the play-field show that a battle is being fought. Trees catching fire, buildings getting destroyed, smoke and rubble everywhere; as the battles go on, it was very important for the project that the battlefield would also show this element. While not everything is destroyable and not catching on fire, creating these elements that the players can destroy accidentally or on purpose creates a sense on consequence and reaction to player actions.

5 TESTING

As the subject of testing is very broad and could hold a thesis on its own, this section only overviews some of the methods that have been used and explains them briefly. While also going into little more in-depth about some of the problems that have been encountered and how they were solved.

5.1 Using AI for testing

It proved invaluable for the project having the AI planes flying around in a very early stage of development. This created the possibility that everything developed that somehow related in-game was able to be tested "on the fly". The AI also cheats as it can act on information instantly rather than taking its time to think like humans would do. This helped to find many rather weird bugs in the code that would only happen in very specific situations. One bug that returned many times during the development was the hangar-man Butters, who is always causing trouble (Image 17). Because AI player can generate inputs much faster than human player, sometimes Butters would stay outside the hangar "dancing" and refuse to go in. This would then spawn multiple instances of Butters when a new plane was spawned, and this then creates a memory leak that after time will crash the game because there would be n number of Butters dancing on the airfield.

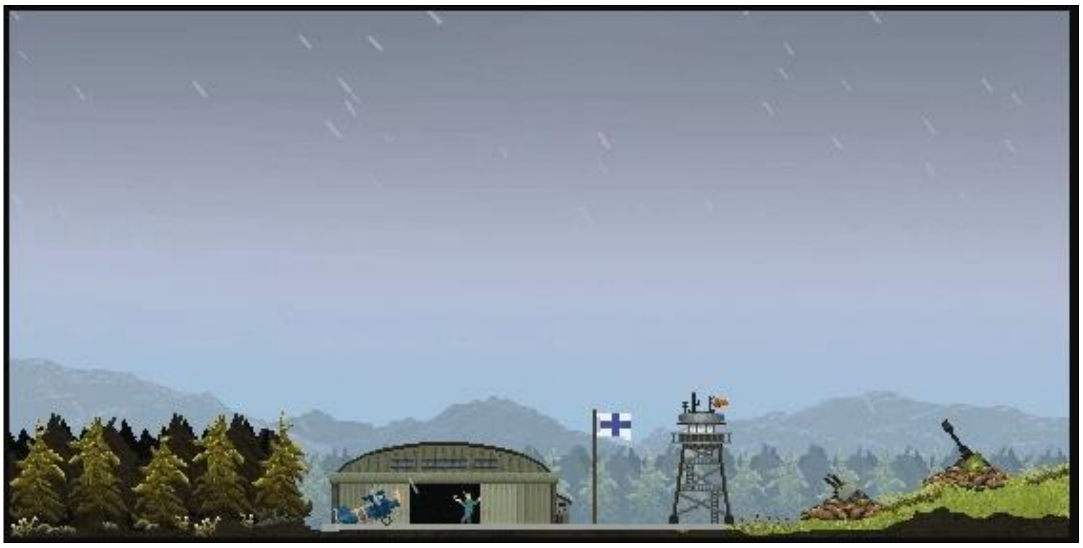


Image 17. Butters on a lunch break.

Play testing

Stability and play testing was the main work for AI players. It was common during development to leave AI battling against each other for multiple hours at a time to check for the overall stability of the game. GMS has a somewhat good error system, so after a crash happens, it will point the developer to the right direction from where to start looking the reason for the crash. Doing these marathon test runs are also very good for checking out memory leaks (Image 18). Sometimes the leak is so small that it is very hard to notice using the GMS provided analytics and developer tools, so running the game while AI plays the game for multitude of hours can enlarge these problems for easy detection.

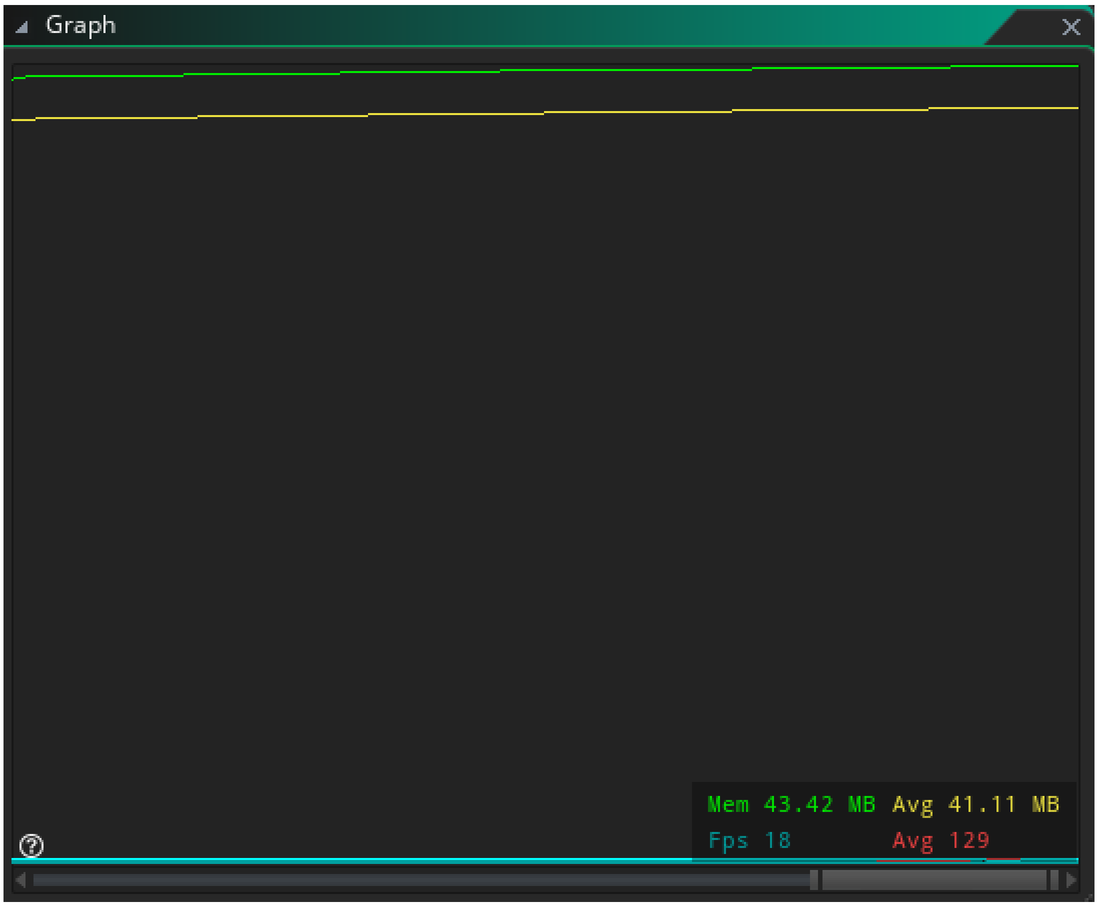


Image 18. Consistently rising memory usage curve indicates a leak.

Infantry

Troopers, like every enemy object in the game have at least some kind of scripted AI like behavior implemented. Troopers have their own behavioral scripts that make them act when certain situation occurs, so after the basic trooper AI was implemented, testing them was more overseeing an automated process than controlling anything. When testing for a specific use-case in the trooper AI, manually controlling a plane and trying to trigger that behavior was the fastest route to achieve results. Trooper AI had to be completely autonomous when patrolling around the battlefield so the only way to achieve this was to just let it test itself, while creating different test situations and taking notes on how it would handle itself.

Anti-Air Defense

Cannons work with same principle of scripted AI like behavior and will always perform the same way under a similar circumstance. It would have been possible to create scripted events for planes to test certain situations repeatedly but during the development there was never a big enough need to build a testing environment to quickly create these scripts and have these specific test cases. Instead manually flying the planes and testing cannon AI in different situations. With the help of AI assisted play testing there is almost 100% certainty that the cannons work as intended. Having a kind of fall back in the code when the AI script does not know what to do, resets itself and will not break everything and crash the game, helps with those rare situations that can sometimes occur when dealing with multiple planes coming in and out of cannon range and dying in weird ways.

5.2 Using humans for testing

Throughout the development humans have been the projects greatest asset in testing the game and getting that valuable feedback for what works and what does not. Mostly the game was plaid by the main developing team, with almost in 1:1 ratio of play testing and development hours. From the very first prototype, every change in code, minor adjustment to gameplay to major gameplay modes, everything had to be tested. There is a certain way how the game should feel, and what feels good and what doesn't, it is easy to test and say if the change made is good or not. Also having multiple opinions on a feature helps to shape it in the right direction as there is no need to trust single developers' abilities alone. After countless hours of testing and playing the game, ones' opinion of the game and its mechanics can become little biased.

Outsourcing play testing and feature-testing to trusted partners gives unbiased feedback on gameplay and mechanics. MansikkaMarmeladi.io has been regularly inviting them to playtest the overall game and sometimes just single features. Not giving any hints or tutoring how things work or what is to be expected to happen gives an opportunity to learn by watching how real player would act. People that have been regularly used to test gameplay are also very good giving feedback on how things feel compared to last time they played the game. Addition to just testing the game, seeing someone enjoy the game is a great test to the game itself and that it is fun to play (Image 19).

Testing the fun factor and stability of the game in larger scale, a demo was also released that features 1 skirmish map playable with AI or human players. Demo did not give any meaningful data that would have helped in the development or validation of the game. This was kind of shame but expected. Demo was released in itch.io that is the go-to place for indie developers to share their games. Problem was that the demo did not attract enough audience to get a good amount of feedback about the game or how it plays. Best feedback with new players was gathered during demo days and special events that will be presented more widely later.

Image 19. Outsourcing playtesting.

5.3 Importance of different test hardware

At the company's disposal with minor modifications there are over 14 different kinds of computers that vary greatly in performance, screen resolution, age and operating system. This is a great resource to be able to test the game in-house without spending time on gathering feedback on different types of guest-machines. Almost all the computers are Windows machines from different eras from Windows 7 to Windows 10; Windows is the main target platform for the game.

Benchmarking started very early on development. Low performance machines had the ability to spot bottlenecks in the game very quickly and from there it was easy to set a point of reference on how the game should be able to perform with low-end computers. When the game grew larger and more complicated, the need for performance also grew or at least this was expected. Further inspection of the code showed some poorly optimized design choices which made the game run awful with those low-end machines. The game is drawn only in 2D, so it should be able to run with pretty much anything that has a decent modern not more than 10 years old integrated graphics card. If the user has a dedicated GPU, game should be always performing well. With some optimization and code refactoring, benchmarking was back on track.

During the development there was more problems related to low performance machines than high performance machines but that does not mean high performance machines passed without troubles. The game time or tick is based on a 60 frames-per-second clock, so one tick is 1/60th of a second. This was the easiest way to build the game with GMS engine. This means that if the game cannot achieve the desired 60-fps clock speed, it will run slower. This is not as much a problem as if the game wants to run over the 60-fps clock speed, this could be the case when user has a monitor capable refreshing the screen over that 60-fps limit. Current gaming monitors can go up to 240Hz and luckily company had 144Hz screen that could be used for testing. The problem occurs when the game is forced to run in-sync with the desktop refresh rate, in this case 144Hz. So, everything in the game world is 2.4 times faster. This problem could have been avoided by building the game time based on actual time and the refresh rate of the user, but this problem never occurred in the beginning of the project. GMS can deal with these scenarios with the current system, but this needs the developer to be aware of the problem first as GMS cannot handle this kind of problem automatically.

Sometimes weird, possibly hardware related problems have occurred in the test machines. There was this very consistent problem with one of the laptops that every 30 seconds there would be a big lag spike that would halt the game completely for a few moments and every 8 seconds a minor spike would occur. After the optimizations mentioned above were implemented in the game, lag spikes were not so defining but still occurring consistently and only with this one laptop. This was suspected to be caused by graphical memory buffer running out of space or something related as it is happening in consistent intervals, but this theory was never proven, or any evidence found even with rigorous testing. It is plausible that this problem will come haunting after the release with more test hardware available which means the buyers of the game.

The game is planned to be released on all three main platforms; Windows, Mac and Linux. The problem with both Linux and Mac release is that each platform needs the game to be compiled separately on the target platform, for the target platform. In short this means that when compiling the game in a Windows machine, it cannot run in either platform; Linux or Mac. With each platform come platform specific problems; like controllers not recognizing correctly or the code is running in wrong order, so things will not work as expected. There has not been any testing done on either system, so at this point it is impossible to say if there are any problems but the safe bet would be definite yes. With the limited resources available for the development, if running and testing the game in either system turns out to be too much of a hassle, both version will skip the initial release and will be adhered later.

5.4 Creating test data for analyzing

In web development it is common to use heat maps to gather data from the users using the site. Heat map is drawn as an overlay for the site administrator that shows how the users have interacted with the site, where the mouse cursor is and how it moves during the visit and what content is clicked (Image 20). In game design heat maps can be used to visualize different types of information, mostly to identify imbalances in the current map or scenario. (Gamasutra - Balance and Flow maps). The principle works the same way as with web development, user interacts with the game while the game collects the data that can be later used for an example; heat maps.

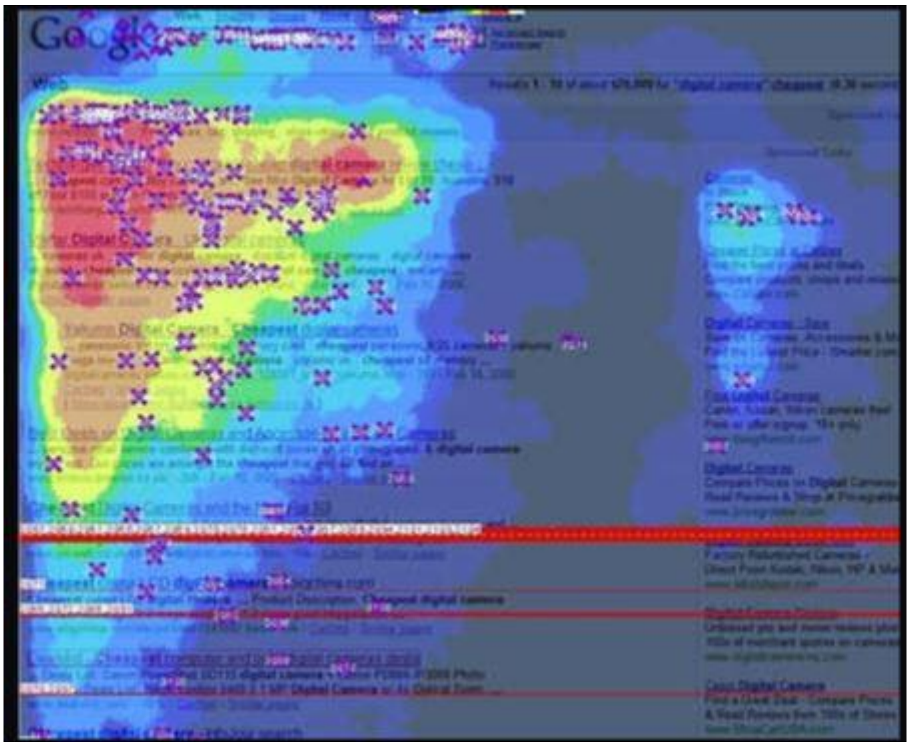


Image 20. "Google Golden Triangle by Amit Agarwal, on Flickr"

A modified version of the heat map principle was created that collects the location of the plane and its color when it is killed. Variable that also gets the object that killed the plane can also be attached to the data. This can be tracked during live gameplay by toggling the option to draw colored crosses on the map in those locations where the deaths have occurred, or while running a long test session with AI which gives an image with the map that has the death locations drawn into it. This data can then be used to determine multiple types of information.

When doing session with AI only, the heat map can show if a certain AI gets killed multiple times in the same location. This can indicate a problem in the AI or just an issue in level-design. If a human player gets killed multiple times in the same spot, then this indicates that the problem is in level-design. For example, certain AA-gun can be placed in such a way that AI and most human players cannot avoid it. This can lead to unnecessary frustration with the player and that is the opposite of what the game is trying to achieve. Solution would be to remove the AA-gun or just place it in a new location.

Data gathered from the deaths heat map can also be used with completely opposite usage; to see places in the level that get no deaths at all. This could indicate a safe spot for players to give some breathing room during a battle. Safe airspaces are generally rare as the game is very fast paced and action tends to "seek the players" but levels with parts where there is no AA insight tend to be bit safer. These spots also give the players an opportunity to dogfight head-on where usually the pilot with superior skills wins the fight.

6 VALIDATION

6.1 Finalizing physics

After countless hours spend playing the game, it never felt quite right. While the game became familiar how it works and plays, there is always room for improvement.

Bombs 2.0

Bombs never felt quite right; there were some trick shots that were not possible with the current physics system. For example, a trick when plane is angled upwards with low medium speed then just at the last second pulling up, stalling the plane and throwing the bomb 90 degrees upwards. When the plane is stalling it dives down and after player recovers from the stall player should be able to fly under the bomb before it drops. This was the sum of two things that needed adjustment to make things work as designed. First adjusting the bomb fall gravity in downwards angle a bit lower, this makes it "hang in the air". Second adjustment was increasing the gravity or fall speed when the plane is stalling. As a result, the trick shot was working as designed and as a side-effect; bombs got a little longer flight trajectory that increased the fun factor.

Speed 2.0

Speed of the planes also turned out to be slightly problematic, as the overall speed of the planes was good but the low amount of drag and strong acceleration made things slightly too fast. The amount of bombs each plane carried also did not affect enough in overall stats of the plane. As an example, the blue plane that is the fastest one with highest acceleration and can carry up to 3 bombs (max is 6), could climb almost vertically with max load, meaning all bombs, ammo and fuel. Compared to the weakest powered plane, the yellow plane with bomb capacity of 6 and lowest acceleration and max speed it would only stall when trying climb vertically with max load. Both planes with their minimum loads could climb vertically with ease and planes just do not do that. It should work in a way that plane needs to build a small amount of speed by diving and then it could climb vertically without stalling, heavier the load, less time it should be able to climb before stalling. Tinkering with the weights and the effect it has on dragging the plane somewhat fixed the issue, but because the blue plane is so fast it also the problem of stopping the plane in time. With minimal load and pulling the plane up engine off, drag could not affect it enough before the plane would hit "skyroof" that marks the top limit of the map and makes the plane spin out of control to its death. This made turning the plane insanely difficult in certain situations. Adding the amount of drag plane has on certain ranges of upwards angle helped a lot with the issue.

All the adjustments affect more on the high-level player as they start to get a knack on how the planes control, it will be interesting to see when there is a solid player-base and those new high-level players start suggesting how planes should control. Now the adjustments are based mostly off development teams’ own experiences and only a fraction on gathered test data from play testers.

6.2 Balancing gameplay

To keep the gameplay fun for 4-players with different level of skill in a same multiplayer match where all the planes behave and act complete different from each other is an interesting challenge to say the least. Bombs were noticed to for being way too effective method of destroying other planes and the planes machine gun would get very little usage. This leads to matches being a bombing fest and not feeling like a dogfight game it was intended.

The range of the machine gun was buffed on all the planes by hefty amount and also the bullet travel speed was increased. After the change in a chase situation the chasing player has a slim chance to get the kill even with the slowest plane when before the bullets would have never reached the chased plane. This also makes jousting much more interesting when players constantly need to evaluate the route of least damage in certain combat situations with multiple threats. Bombs were accidentally made so that you could destroy them in mid-air with machine gun and a decision was made to keep it as a feature as this added another skill element in the game and eliminates some of the cheap kills player could do with bombs.

All the planes have natural weakness to some plane and are strong against the other. Like the blue/yellow pair; because the blue is so fast the yellow can never catch it if the blue decides to flee but because the yellow can turn so quickly, and blue cannot, in the right place like on a corner of the map yellow can obliterate blue at will. Tweaking all the turn rates and speeds values of the planes has been an ongoing fight throughout the development but with enough situational awareness and knowledge of the planes traits a skilled pilot can win any fight. This is exceptionally important part when it comes to high-level play. No one wants to lose because the plane is just under-powered against one type of plane.

6.3 Events

The greatest platform to collect feedback and see the game performing in a real-life situation has been the events the company has been attending, like demo days, meet-ups and showrooms. Getting to see first-hand how people new to the game interact with it, what features really work and how they take the game overall. Good example of a problem that was encountered this way was how new players had a very hard time trying to take-off with the yellow plane and usually ended up crashing it in the first few seconds. This happened because planes start with all the maximum items they can carry and because yellow plane can carry 6 bombs that make it very heavy and slow so taking-off needs bit more fines. For developers and regular tester this was not a problem as they were very used to mechanics and taking-off with heavy loads. Setting planes to start with balanced loads and indicating with the 'hUI' that player can indeed load more to the plane seemed to fix this issue well (Image 21).

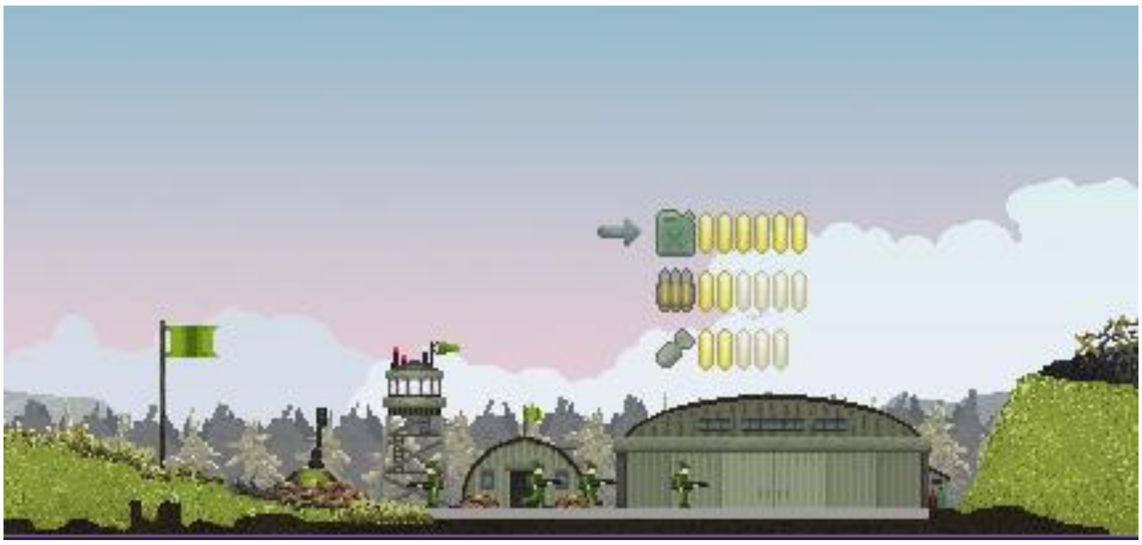


Image 21. Improved hangar UI

Demo-days also showed the great potential game has, when a group of random people can enjoy the game for an hour and even come back later for more. Human players also tend twist the game mechanics to their limits that can reveal unintended mechanics. Like how you can land the plane like a helicopter if correct set of moves is executed. These high-level of play mechanics give sense of accomplishment to those who sink the hours in and take the time to learn these tricks which in process adds more longevity to the game.

The company attended the first public event in about five months of development time. This gave the necessary feedback to see if the game has any potential at all and if it is worth sinking in more development time. Now almost two years later and the launch around the corner the answer was clear, the game really is fun to play.

7 EPILOGUE

7.1 Further development

As of writing this thesis the development of the game is still an ongoing process. There is at least 5 months' worth of work to be done before the game can be released. AI needs more work the get it operating as intended in every scenario. For example, adding scripted events and other programmable timelines to support the overall gameplay. The progression system is not finished yet as the overall rank/level up mechanic needs to be created and tested and design the time it takes the player to go from level 1 to max level. The game also needs to be polished to look presentable and for capturing a larger audience.

7.2 Missing components

The solo campaign mode has not been implemented at all at this point. The story has been written and all the supporting elements are complete, for example designing the levels and events and how the missions should play out. However, all the art assets need to be drawn and the actual code to be written, one can just hope that the development will not be encountering any major complications as the solo mode is a major part of the whole experience.

7.3 Final thoughts

This game has been a major part of the author's free time for a long time. One could think that writing this last chapter should be easy after thousands of hours spend on design and actual development time of the game, but it is hard to reflect on it. The development of this game has had multiple long breaks, writing this thesis was the second longest time for the author have taken off from developing the game and only time the author has opened the editor was to take some screenshots of the game for this thesis.

While game development has been strictly a hobby for the author, committing to a large project like this one is not easy. It was never hoped that this game would become one of those big titles that would make considerable sums of money. When first time really committing to this project it was agreed that if breaking even not counting in the time that has been put in, game would be deemed success. Even if this has been just a hobby, after all the time spend it would be nice to see it bear some fruit also. However, as the game industry is ever growing, and one really must fight for survival and visibility in this flooded and saturated market, making any profit is mostly just a game of chance.

One lesson that was really learned during this time is that one should never start a game company if one’s survival would depend on it. Only with financial freedom could a person really think to work just in the game industry and not doing anything else as ones’ day job. The authors respect goes out to the people who have made the name for themselves in this competitive business.

The final product can be seen at the commissioning company’s official page. (http://mansikkamarmeladi.io)

REFERENCES

Chandler, H. 2013. The Game Production Handbook. USA: Jones & Bartlett Publishers.

forbes.com - Crash Bandicoot. Referenced 24.9.2018
https://www.forbes.com/sites/erikkain/2017/09/24/crash-bandicoot-n-sane-trilogy-has-sold-over-2-5-million-copies-on-ps4/#2ff067625a70

Gamastura - Balance and Flow maps, Referenced 22.10.2018
https://www.gamasutra.com/view/news/125213/Opinion_Balance_and_Flow_Maps.php

gamesindustry.biz - YoYo Games. Referenced 25.9.2018
https://www.gamesindustry.biz/articles/2017-03-08-yoyo-games-our-competition-with-unity-is-all-in-peoples-heads

gamesindustry.biz - YoYo Games. Referenced 25.9.2018
https://www.gamesindustry.biz/articles/2017-03-08-yoyo-games-our-competition-with-unity-is-all-in-peoples-heads

github.com - Triplane Turmoil. Referenced 24.9.2018
https://github.com/vranki/triplane

GML Manual - Data structures. Referenced 26.9.2018
https://docs.yoyogames.com/source/dadiospice/002_reference/data%20structures/index.html

GML Manual - 'with'. Referenced 26.9.2018
https://docs.yoyogames.com/source/dadiospice/002_reference/001_gml%20language%20overview/401_18_with.html

Google Golden Triangle by Amit Agarwal, on Flickr
https://www.flickr.com/photos/amit-agarwal/2052668047

Rantanen, S. 2017. AI for warplanes in a 2D side-scroller.
http://urn.fi/URN:NBN:fi:amk-2017091815174

websitetooltester.com - comparing 2D engines. Referenced 25.9.2018
https://www.websitetooltester.com/en/blog/best-game-engine/