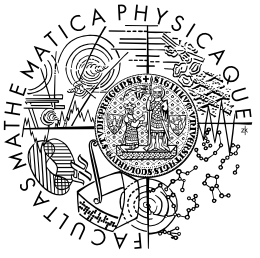

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jiří Malý

Real-time Strategy game toolkit for Godot engine

Department of software and computer science education

Supervisor of the bachelor thesis: Vojtěch Černý

Study programme: Computer Graphics, Vision and Game Development Bc. (NIPGVAVH19B)

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Title: Real-time Strategy game toolkit for Godot engine

Author: Jiří Malý

Department: Department of software and computer science education

Supervisor: Vojtěch Černý, Department of software and computer science education

Abstract: Real-time strategy (RTS) games have been on the rise in recent years, but the tools for making them are scarce. This thesis presents a toolkit for creating 2D RTS games in Godot engine and the language of choice is C#. The toolkit contains templates for units, buildings, abilities, attacks, human and AI players, maps, menus and user interface along with examples of their use in a demo game. Both user and programmer documentation is provided and the toolkit is open source and thus allowing for maximum modularity. Using the toolkit allows the user to start developing a game or making a prototype straight away skipping the technical minutiae at the start.

Keywords: RTS, Godot Engine, Real-time strategy, Toolkit, Godot Csharp

Název práce: Sada nástrojů pro vývoj realtimových strategií v Godot enginu

Autor: Jiří Malý

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: Mgr. Vojtěch Černý, Katedra softwaru a výuky informatiky

Abstrakt: Real-time strategie (RTS) jsou v posledních letech na vzestupu, ale nástrojů k jejich výrobě je málo. Tato práce představuje sadu nástrojů pro tvorbu 2D RTS her v Godot enginu a zvoleným jazykem je C#. Sada nástrojů obsahuje šablony pro jednotky, budovy, schopnosti, útoky, lidské a AI hráče, mapy, menu a uživatelská rozhraní s příklady jejich použití v demo hře a uživatelskou i programátorskou dokumentací a zároveň jsou open source a umožňují tak maximální modularitu. Použití této sady umožňuje uživateli začít s vývojem hry a prototypováním okamžitě a neztrácet čas s technikáliemi.

Klíčová slova: RTS, Godot Engine, Realtimová strategie, Sada nástrojů, Godot Csharp

Contents

Preface

Real time strategy (RTS)[1] games are a video game genre that not only includes succesful video games but has also spawned multiple other genres from community generated content. The term RTS is not formally defined and so for the purposes of this thesis it only refers to the Warcraft[2]-like games in which the player takes on the role of a commander of his faction's forces having absolute control over individual unit's, the infrastructure producing them, their upgrades and resource management/gathering. Titles such as Warcraft 3 and StarCraft[3][4] have not only been commerecially succesful but the editors shipped alongside them have given rise to the popular Multiplayer online battle arena (MOBA)[5] genre (games such as DOTA 2[6] and League of Legends[7]). However these games and therefore also their editors are now at least a decade old and have always provided only superficial control over the inner workings of their engines since they are closed-source. Attempts at creating open-source RTS engines have been made[8][9], but with little publicity leading to almost no widespread use.

This project's goal is developing a toolkit for making RTS games in a well known Godot engine [10]. It aims to have similar functionality to the editors shipped with RTS games mentioned above such as creation of simple games without having to write much code, thus appealing to non programmers, as well as being open-source just like the engine itself for maximum control over every aspect of the game (appealing to game developers). Having a toolkit such as this one can also help game designers to build a quick prototype to show off even if they wanted to build their own engine later. It would be also useful for attendees of game jams who are time constrained and do not have time to waste with technical minutiae.

Godot engine natively supports GDscript,C# and C++. This project is mostly written in C# because of its type safety that allows to build more solid object-oriented structure than GDscript and is much faster to develop in than C++, because it is less tightly integrated with the engine and thus mistakes are easier to identify and fix.

The goals of this project are: Create a toolkit that has as much functinality as to allow for making a simple RTS without having to write additional scripts, but providing API to write them. Create a demo game that shows off most of the things possible with the toolkit script-less.

1 Background

1.1 Real time strategy description/definition

As with most video game genres and many terms used in the video game industry their meaning is inconsistent and prone to interpretation. Real-time strategy games can be a very broad term where the only fixed features are the "strategy" and the "real-time". Effectively what this means is that any game that is not fully turn based and requires any kind of strategy is technically RTS. The term was coined alongside the marketing for DUNE II [11] which itself drew inspiration from Herzog Zwei[12] and was later used to describe Blizzard's Warcraft[2] and StarCraft[3], Westwood's (EA's) Command&Conquer[13] and others. (Note to avoid confusion: Warcraft refers to the Warcraft trilogy(1994,1995,2002) not World of Warcraft(2004)[14] (which is an MMORPG[15])) Warcraft 3's editor was due to the game's popularity heavily used by modders to create not only maps and campaigns but also games that used the editor to create or adopt other genres. The map for Warcraft 3 called Defense of the Ancients (DOTA)[16] gave rise to the MOBA[5] genre and this is not the only case of mod/map creating a new genre. Thus in a sense all MOBA's and the other derived genres could be technically called a subgenre of RTS. The toolkit should focus on RTS games and the tools should be made with RTS in mind, but they should not be limiting if a user had a different genre in mind. And so lets focus on following pillars of RTS:[17]

1.1.1 Resource management

What separates RTS from Real-time tactics (RTT)[18] is base building and resource management. The player often starts with a set amount of resources or has some way of aquiring more. Warcraft and StarCraft have sources of resources placed in strategic locations around the game map and in each player's starting area. Their workers (designated unit type) are then commanded to patrol between the source and a player owned building, each trip bringing the player a small amount of resource. Since the production of workers costs resources this forces the player to decide whether they want to first saturate (have the most efficient amount of workers gathering resource) and have stronger economy, or if they will first focus on the military to be more aggressive at the cost of being in economical disadvantage. Workers also often take up space for military units meaning that player with more workers will have stronger economy but can never have an army of an equal size to a player with less workers. Depending on often vulnerable workers for the supply of resources means the player is incentivized to protect them and sabotage opponent's.
Company of Heroes[19], a game more focused on combat and less on the economy instead has steady trickle of resources based on the amount of territory player controls. This means that the player is entirely focused on expansion and this is a reinforcement loop pushing the player always forward.
RTS games often have only a few resource types: Warcraft and StarCraft have 2, Company of Heroes has 3, Spellforce [20] has 4. And they often have different

use and means of acquiring. In general one resource is the fundamental one and workers (if the game has them) usually cost this resource, the other resources are used for more advanced units etc. In StarCraft the secondary resource is acquired at slower rate (cca 1/3 of the speed of gathering the primary resource) and needs an investment of the primary resource to even be gatherable. Early units and buildings often cost only the primary resource and the more advanced a unit is the more its cost increases and shifts toward the secondary resource. This means that if a player wants to be most efficient with their resources then they should focus on the middle of the road units to spend about equal of both resources, or have a few advanced ones supported by basic ones - giving a variety to the strategies.
Company of heroes has a primary resource and two secondary ones, used for unit abilities and advanced units respectively. Some spots on the map generate high volumes of a secondary resource meaning that if a player wants to have it or deny the enemy from having it they should focus on capturing and defending the territory. This gives the players reasons to fight even in unfavorable terrains giving variety to the combat.
Spellforce splits the resources into those more used for buildings, those used primarily on units and those used for advanced technology. And while StarCraft and Warcraft often have both resources present at a a specific point on the map Spellforce is more keen on having the player build bases dedicated to a specific resource.
Having more than 4 resources often means the player needs to focus on the city-building aspect more. For example Ubisoft's Settlers [21] can have dozens of resources, where some are found on the map, others are produced by certain buildings, advanced resources are refined from the basic ones and elite ones are refined from the advanced. This means the player is focusing mostly on the logistics and balancing the production of all the resources and less on combat.
The toolkit should therefore support any amount of resources allowing for any flavor of RTS.

1.1.2 Units

The control of individual units is a staple of RTS genre. It is what differentiates it from city building simulators[22] like SimCity [23]. Units are owned by a player and in vast majority of cases can be only issued commands by the player who owns them. Universal commands are: move, stop, hold position and attack, but units can also have special abilities. The player often has a selection of unit types they can produce. Each unit type has its own statistics - health, attacks it can perform, special abilities it can do, its movement speed etc. Units are most often produced from buildings (at the cost of resources) but it is not unheard of for units producing other units or there being some abilities capable of summoning them even for free.
RTSes often have some kind of squad movement - if multiple units are issued a move command they arrange themselves to a formation to march. In case of Company of Heroes the player is producing and controlling squads of units (unable to command a specific unit in the squad on its own) and the squads are always moving together. TotalWar [24] games even went so far as to control entire regiments of dozens of units. However squad movement is often a closely guarded

secret of the companies making these games and implementing it would be a paper on its own.
RTS games often have campaigns that follow the story of a character in the world the game takes place in. This sometimes leads to heroic units and the merge of RTS with RPG [25]. These are unique units that often represent the campaign's protagonist and other important characters and their power level often reflects this status. In some games the death of this "Hero unit" means failing the mission, but most often there is some respawn mechanic often tied to a player owned building or a structure on the map.
There will be no explicit implementation of Hero units in the toolkit, but it should support creating them.
The toolkit should therefore provide the basic structure for a unit (movement, health, graphics) and allow for creating own units with user defined statistics.

1.1.3 Buildings

Buildings are the backbone of the player's economy, are responsible for rebuilding the army after combat and are the place where the player has their last stand. Their importance depends on how much is the game focused on combat vs. citybuilding, but the toolkit should be able to accommodate either.

1.1.4 Graphics

Most modern RTS games are 3D, but only for visual purposes. The player perceives the world from birds eye view and thus the game world has a few vertical layers, but those can - and in 2D RTS are - easy to simulate without the need for 3D. StarCraft 1 for example has verticality and flying units but is a 2D game.
That said there are 3D RTS such as Homeworld[26], which takes place in space and therefore can make use of the 3rd dimension.
However the computer mouse and monitor are 2D devices and thus the games are still limited by the fact that the player can interact only with a cross section of the game world at a time. In games in which players control only one character this is not a big issue since the camera's position is locked to the character's position and the player controls only the camera's rotation. In turn-based games the player has the time to orient themselves, because they are not pushed by the need to respond in real time. RTS requires quick responses on various places of the map for the entirety of the game and so simplifying to 2D helps with the game being far less overwhelming.
The toolkit is made for 2D as it is the easiest to use for new developers and games that use the 3rd dimension work on a technical level very differently from the classic RTS and would require completely different approach making case for 3D RTS toolkit rather than being part of a 2D one.

1.2 Godot Engine

Godot engine[10] development started in 2001 and as an open source project first released in 2014 and is one of the top 3 engines for indie development[27] alongside Unreal[28] and Unity[29]. Unlike the other two it is open source under

the MIT[30] license. Despite being almost 10 years old not many successful games have been made in Godot and is used mainly for small projects, prototypes and Game Jams. It started to become relevant with its version 3.0 released in 2018 which addressed a lot of issues stemming from its closed-source development and due to Microsoft donation introduced C# as a scripting language[31]. In 2023 version 4.0 was released which focused on Vulkan support and multi-core processors. And due to Unity's controversial licensing announcements in September 2023 Godot received multiple donations to turn it into a viable alternative.
Godot engine was used to make successful games such as Dome Keeper[32], Halls of Torment[33] and it has been announced that the sequel[34] to popular roguelike deckbuilder Slay the Spire [35] is being developed in Godot too. Godot 4.2 is as of the time of writing the latest stable version of the engine and is the one used to create the toolkit featured in this thesis.[36]

1.3 Programming/scripting languages for Godot:

Since the engine itself is written in C/C++ all supported languages essentially just call the engine's C/C++ functions through APIs. Languages available in the Godot engine:

1.3.1 Natively supported languages

GDScript Object-oriented programming language with gradual typing (Typescript-esque) and is the number one language for scripting in Godot. It is inspired by Python (Python was the original scripting language for early versions of the engine)

C# Contrary to GDScript is statically typed and thus allows for more solid object-oriented design. And since Godot 4.0 the language is nearly on par with GDScript with what it can do.

C/C++ can be used either by directly editing the engine itself (requiring recompilation) or via GDExtensions. A GDExtensions project produces a dynamic library and thus it doesn't need to recompile the rest of the engine while acting almost like part of the engine itself (some of the deepest functionalities cannot be accessed this way but that is beyond the scope of anything needed for this project) Nodes created via GDExtensions can be extended only via GDScript (glue-code for C# is not generated alongside as of the time of writing).

1.3.2 Other languages & Language interactions

Some 3rd party languages (Python, Rust, D, Nim, Go etc.) can be used with Godot, but their support is community maintained and unofficial. C# code and GDScript code can interact (call each others functions etc.),but since C# is more type constrictive, calling C# from GDScript is easier than vice versa.

1.3.3 Use of languages within the project

In this project the main structure and game logic is written in C# with small parts moved to GDExtensions (C++). Ideally the project would be written in C++ in its entirety, however because of the lack of documentation, inability to pair with C# properly[37], longer compile times and worse debugging experience it is more convenient to wait until Godot improves these and in the meantime use C#
There are some disadvantages with using C# over GDScript. The documentation is slightly worse and the userbase and therefore online help is focused on GDScript. Thankfully the GDScript code is not much different from the C# one and if a solution exists for GDScript it often takes just a brief consultation with the documentation to find equivalent methods, types etc. However there are features that are only available for GDScript such as descriptions for exported variables which would be extremely beneficial in the context of making a toolkit to make the experience more user friendly. Right now it is limited to code comments and external documentation. Some C# features also don't quite translate to Godot. For example it is possible to declare Signals in an Interface without causing any errors, but also without any effect. Since Signals are not a part of C# they are generated by Godot, but in the case of Interfaces this functionality doesn't work as expected. It has to be worked around by enforcing methods through the Interface that get called when the signal is triggered and the programmer has to know they have to implement said Signals alongside these functions.

1.3.4 Code structure

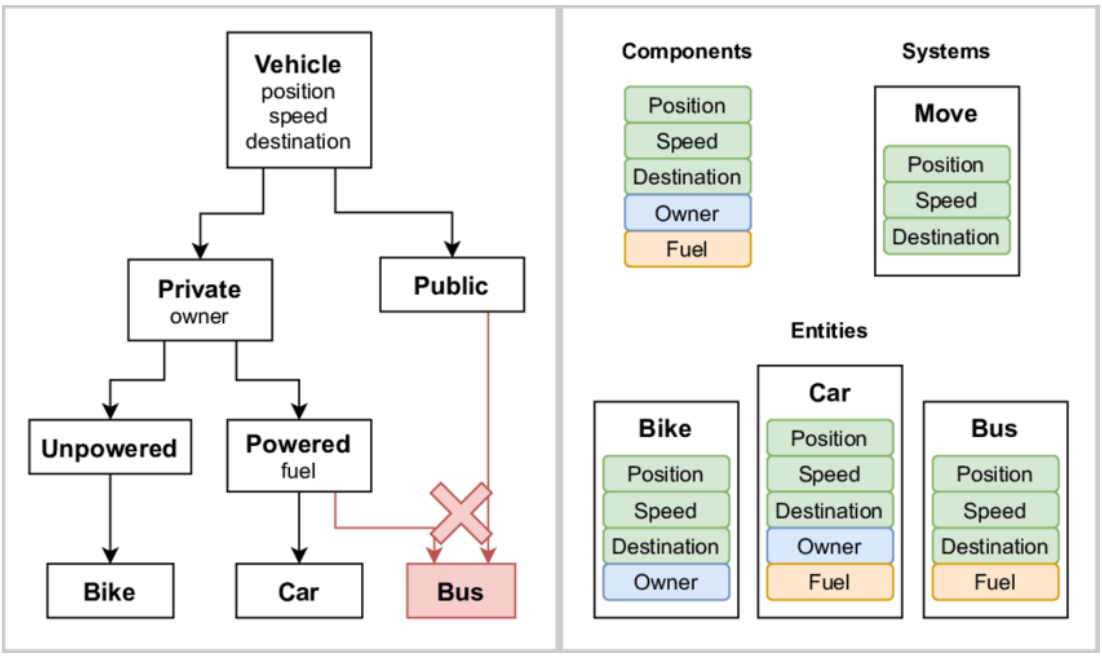

Figure 1.1 Object Oriented Programming (OOP) vs Entity Component System (ECS)[1]

[1]https://www.researchgate.net/figure/Object-Oriented-Programming-OOP-vs-Entity-Component-System-ECS_fig2_352242590

In game development there are two often debated ways to structure code: inheritance and composition [38]. Inheritance is the primary principle used by OOP, but it also has many critics. In game development composition is often used as an alternative to inheritance and is the foundation of ECS architectural pattern. It is important to note that while many articles discuss "OOP vs ECS" what they actually discuss is inheritance vs composition, as OOP is a programming paradigm while ECS is an architectural pattern. In fact ECS is OOP based as its entities and components are technically objects with their own methods and data. Therefore it is necessary to specify that by comparing OOP with ECS we compare inheritance with composition.

OOP Object oriented programming is a programming paradigm [39] - high level way to structure a program that bundles related code and data into objects. This is especially useful for game development, because it aligns with the structure of the games themselves. For example: enemy is an object with code that allows it to attack the player and data that store its statistics. The most common way to then structure the code is via inheritance - having a general purpose class that is later specialized and expanded on, creating a forest structure (multiple trees) in the code. The positive of this approach is that every object can be tracked back to its parent classes and code that works on parents (unless specified differently) works on the child classes as well. Problems start to arise when the programmer needs the functionality from two branches of the same tree - creating a cycle in the tree. What this means is that the child inherits the common ancestor of its parents twice, creating ambiguity when calling these duplicated methods and taking more space in memory because of duplicate data. This is called the diamond problem [40]. And each language deals with this problem differently:

- C# uses interfaces - a special type of fully abstract class that contains only declarations and no definitions. If there is a functionality that doesn't fit the vertical tree structure it can be specified via interface. The definition then has to be written for each class that implements the interface so there could be some code duplication, but it allows for multiple classes of different ancestry to have the same methods without having to result to duck typing (which is worse performance-wise and not type safe which goes against the spirit of C#)
- GDScript solves this issue mostly by duck typing because it is not strongly typed, so if an object has said method it can be called despite the objects sharing no ancestry.
- C++ allows multiple inheritance but the objects now have duplicate code and the programmer has to ensure there is no ambiguity when calling the duplicated methods. C++ can also do duck typing but if we use C++ we are after performance and using duck typing would be counterproductive.

ECS Entity component system is a software architectural pattern that uses entities and components and follows the principle of composition over inheritance. In short entities are similar to the classes at the top of inheritance hierarchy - small in size and very generic, but instead of creating inherited child classes,

new separate classes are created that implement the required functionalities, called components. The final object is an entity that has one or more components attached to it with only a small piece of code that glues them together. In practice this glue is often a class that inherits the entity class but contains very little code. This is incredibly useful when there is a lot of features that can extend the parent class that needs to be combined. With inheritance only, this would result either in huge objects with multiple flags disabling certain functionality or by having a lot of duplicated code across multiple inherited children. In essence ECS is simply a compression algorithm - instead of having large children with duplicate code this duplicate code is pulled into separate objects (components). ECS is especially popular in game development because game engines are generally made with it in mind and make adding components to entities very user-friendly.

It is important to capitalize on the benefits of both of these pattern's strengths - the core of the toolkit just like Godot itself is built with inheritance, but the unit types and other gameplay classes will be entities extended with components. In Godot the Object class is inherited by nearly every other class in the engine, notably Node and Resource which are the two pillars of Godot's gameplay related code. All other classes that the user comes into contact in the editor are created through inheritance while adding children to Nodes in the inspector is prime example of component approach.

1.4 Requirements

RTS is an old genre (32 years since the release of Dune II at the time of writing) and thus a lot of mechanics have over the years converged into a core common for almost any RTS. If we are making a toolkit we should focus on pinpointing this solid core so that when a user makes a least effort game it should be mechanically similar to a generic RTS. Let us try to pinpoint the core mechanics:

1.4.1 Controls

The game requires the mouse&keyboard combination. Rarely can an RTS be played with a controller as it doesn't lend itself well to the gameplay. It is a standard that an RTS should be playable with just the mouse - any irreplaceable command or feature should be part of the User Interface (UI) as a button. This allows for casual gameplay and also shows all the options available to the player on the screen. It is much better for new players to start with using just their mouse, seeing the description of every action they can do on button tooltips and only once they start feeling bottlenecked by the mouse can they switch to key shortcuts. There are even players that play competitively and have thousands of hours in RTSes that have never used a keyboard. This flexibility of choosing whether to use keyboard or not is a part of strategy (not just real-time) games in general and it lends itself well to the "armchair general" fantasy many of these games want to capture.
There is one mostly keyboard specific core mechanic and that is saving control groups - once a player selects a group of units they can bind it to the top row

number keys to select it quickly again. Some newer RTS games made it part of the UI, which was not common with the older titles most likely due to the UI already taking a lot of space on the screen because low resolutions in those years required larger fonts to be legible leading to by current standarts unnecessarly big UIs.

1.4.2 Camera

The player should have a bird's eye view of a section of the map. Camera is not locked to any character or position and should be moved by moving the mouse cursor to the sides of the screen - pushing the view in that direction. Occasionally camera movement is also bound to the arrow keys but since mouse is mandatory and the keyboard hand is meant to be near the WSAD keys it is not a required binding.

1.4.3 Selection & Selectables

Almost anything interactable (that is part of the game world and not the UI) in RTS is also selectable (exceptions are few and far between - for example some resource nodes or demolition charges in Company of Heroes 2 that explode immediately upon clicking them).
Note: Selectable is a term used in this thesis to describe an object that can be selected and in the toolkit it is a name of a class with the same functionality, but it is not a common term used in RTS.
Selecting means:

1. Highlighting it in the game world - usually by outline or ring under the Selectable colored accordingly if it is owned (green/blue), neutral/allied(yellow) or enemy(red) (colors are universally like this but not set in stone).

2. Displaying its available abilities in the bottom right part of the screen.

3. Displaying its statistics on the bottom of the screen if it is the only thing selected.

4. Displaying list of selected Selectables on the bottom of the screen if there is more than 1 Selectable selected.

5. Displaying a unit portrait - often animated graphic depicting the selected unit.

1.4.4 User Interface (UI)

UI is a critical part of RTS gameplay. It often covers little less than 10% on the top of the screen and around 25% on the bottom. It is however not unusual to have the top UI removed and its elements incorporated to the bottom bar to obstruct less of the player's view and put all gameplay relevant information in one area. UI often consists of the following:

- Menus and other non-gameplay related elements are usually located on the top left

- Resources in the top right.
- Minimap on the bottom left
- Selection related UI in the center and right side of the bottom bar.

Godot has very flexible UI settings so despite this being the default it is not difficult to move the UI elements around to suit the toolkit user's needs.

1.5 Project Goals

The main goal of this thesis is to create an open-source toolkit for Godot engine to make 2D RTS games.

1. Conceptual goals
   (a) The toolkit should contain a basic AI player that can to a degree play a game made with the toolkit.
   (b) The toolkit should allow to create a simple game without scripting.
   (c) There has to be a demo game made using this toolkit.
      i. The demo has to show all implemented mechanics.
      ii. The demo has to be a playable game.
   (d) The toolkit should direct the user to the core RTS mechanics outlined in the Requirements section above.
   (e) The toolkit has to be extendable as much as possible.
2. Technical goals
   (a) The addition of AI players, new units, buildings and abilities should be as easy as possible to implement simply through C# 's inheritance.
   (b) The toolkit should be OOP and component-based friendly - Things can be extended via inheritance and via components.

2 Implementation

2.1 Development documentation

The toolkit started with Godot v3.5 and was later moved to Godot v4.0 and finally to Godot 4.2 specifically with the mono build (so that we can use C#) from Godot's official website.

2.1.1 Program structure

The C# project is separated into namespaces with the prefix RTS

Physics This namespace holds structs that represent the units of measurement used within the toolkit. Its code is located entirely in Physics.cs. It makes the statistics of game objects human readable as well as keeping them more type safe. This also allows for dynamic changes of the game speed without affecting non-gameplay related effects. The static classes PhysicsConsts and PhysicsValues hold general data used by the physics structs. The toolkit demo game uses 16x16 tiles. If a user wanted to have a different tilesize it can be changed just here without modifying code anywhere else. The toolkit assumes square tiles. PhysicsValues holds the Gamespeed property. Setting it from the inspector allows for dynamic changes in all gameplay processes as it affects the time related structs (Second,Persec and TilesPerSecond) With the exception of Second all physics structs have names only similar to real world units to not be confused with them. Structs implement some of .NET's Numeric interfaces as needed by the toolkit so far. It is encouraged to add more interfaces as the toolkit expands or is utilized by the user. Base Numeric structs in .NET have dozens of numeric interfaces that can be used in generic math, but to avoid bloating the code with unnecessary implementations they are to be implemented on demand.

  - Persec is a unit of frequency, equivalent to real world Hz. Affected by Gamespeed
  - Second is a unit of time. Affected by Gamespeed
  - Tilemeter is a unit of distance. 1 tilemeter is equal to the width/height of 1 tile and thus converts distances in pixels to a gameplay unit. Affected by TILESIZE.
  - TilesPerSecond is a unit of speed. Affected by Gamespeed and TILESIZE

Gameplay Contains gameplay scripts.

Graphics Contains graphics related scripts.

UI Contains scripts for UI elements.

mainspace is the namespace that contained all code before it was moved into the other namespaces. It contains generally useful classes as well as classes that did not fit to other namespaces

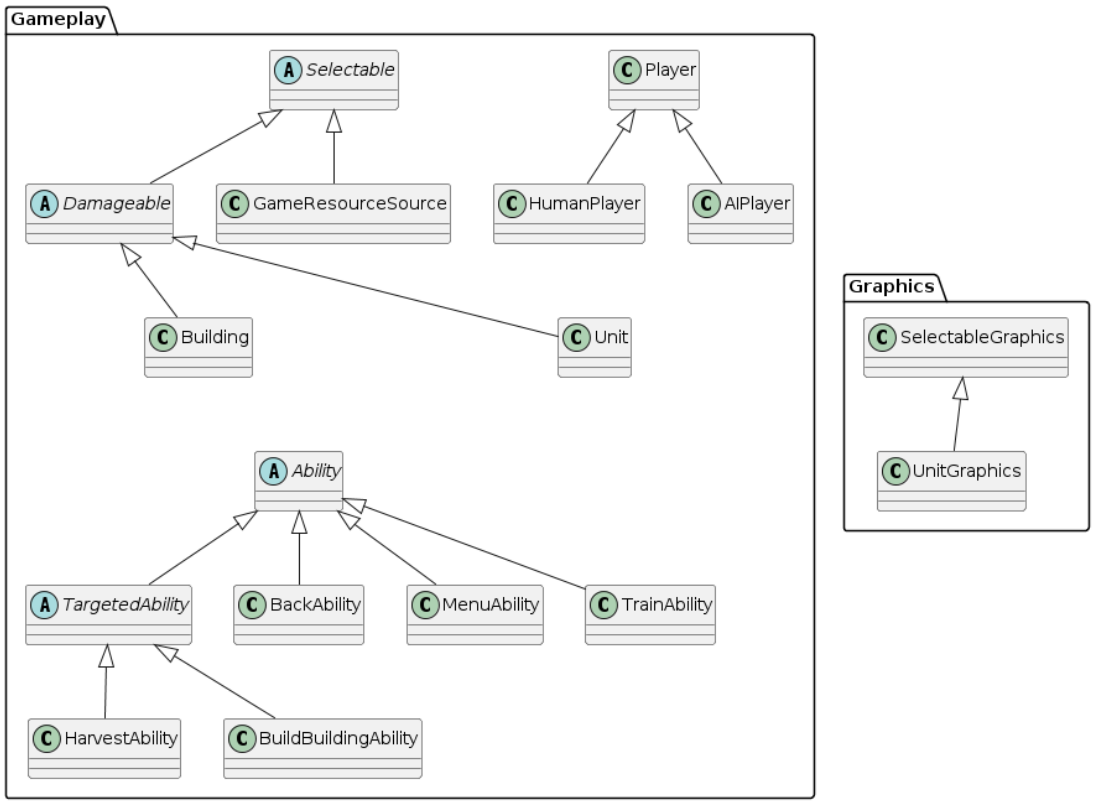

Figure 2.1 Inheritance graph for Gameplay and Graphics namespace

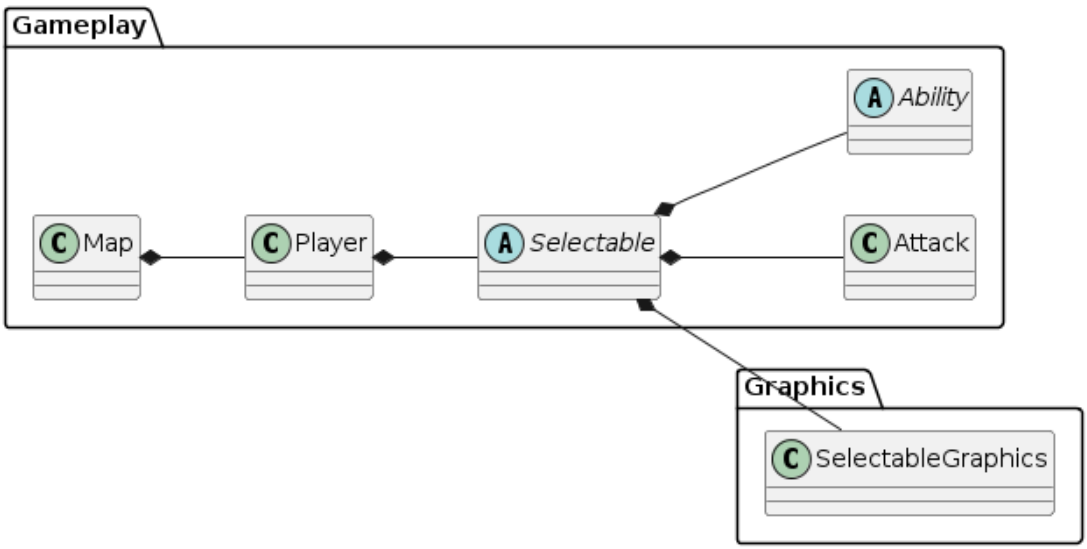

Figure 2.2 Entity component diagram for Gameplay and Graphics namespace

  - GodotExtensions contains static class used to extend Godot's classes and generally useful enums.
  - Menu is a class that handles the main menu and loading of maps.
  - SavingLoading is a class meant to handle saving and loading once implemented. Currently unused and unfinished.

2.1.2 C++ project

The GDExtension project written in C++ is located in a separate repository. The toolkit has the generated .dll from it for Windows x64 for demonstration purposes. To not bloat the repository with dlls for every OS and architecture the other dlls are not included. It is however possible to compile the dlls by following the INSTALL document in the GDExtension project's repository. The project currently provides TileCollisionShape2D node that is used for Building collision and Unit pathfinding around them. Relevant documentation is in the repository also.

2.1.3 Folder structure

In the main project folder contains:

- project.godot file that is opened by the editor
- RTS-Toolkit .csproj and .sln files are there for the C# project. Although .sln is not necessarily required for build it is easier to get started with the toolkit if you are a Visual Studio user and can just run it outright.
- Documentation.md,README.md and INSTALL.md are self-explanatory.
- lib.gdextension and the DLLs folder enables custom C++ library (lib.gdextension registers it with the engine and in DLLs there is the dll itself)
- TODOList.txt is semi-regularly updated list of features that can/should/will be implemented.
- .git* files - used for versioning
- submodule bachelor-thesis -> this thesis repository
- assets folder - contains assets for demo game - scenes, scripts and some resources (unit portraits etc.)
- scripts folder - contains toolkit scripts
- scenes folder - contains toolkit scenes
- resources - contains resources (graphical) used within the project as well as information about their licenses (CC in all cases).

2.1.4 Notes on Attacks and Abilities

Attacks and Abilities are very similar - components attached on a Selectable entity, but they use different approaches.
Attacks are added to the scene tree as nodes. They are a scene that gets instantiated in the editor. This makes it easier to create a new Attack then a new Ability, but the tradeoff is that it is slightly less maintainable - the user must remember to add Attacks to a specific node on the Selectable's scene tree and no other node. This means that there are no compile-time checks and incorrect placement would be discovered only at runtime. This can be remedied by implementing editor scripts in the future. This approach was chosen because Attacks are expected to be far less complicated than Abilities. Where Abilities are expected to often use custom scripts and checking types at compile time might be useful, Attacks are expected to reuse only a few scripts if not having just a single one with flags for enabling features and so strict checking might not be necessary over ease of implementation.
Abilities on the other hand are added through the inspector and require separate Resource for each Ability to enable its addition to the scene tree during loading. Normally the PackedScene resource handles addition of Nodes through the inspector this way, but they are a) not strongly typed and b) cannot have parameterized constructors so a small custom Resource for initialization is used.
These approaches have their advantages and disadvantages. Attacks are much easier to add - simply hanged into the scene tree. This however means that if anything else gets hanged in the place designated to attacks the game will crash or work incorrectly when it tries to use a node that is not of type Attack. Abilities on the other hand are the only thing that can be added in the array in the inspector and export only the required parameters.
Unlike Attacks, Abilities might have multiple things that need to be specified so they can work properly. When adding a resource into a scene tree it automatically displays its exported fields and the developer can immediately see what is required to add.
Note that it would be preferred to use a dictionary instead of array of pairs for the ExportedAbilities property, but Godot dictionaries are not working properly in the inspector yet.

2.2 User documentation

Should the reader be interested in making their own game by using the toolkit it would be recommended to first try to modify the demo game to familiarize themselves with the toolkit's structure, or at least use it as a reference and an aid for debugging purposes.
Following sections describe how to make individual components of a game.

2.2.1 Attack

To make an attack one can copy or inherit from the AttackAnim scene. The root has properties that can be set in the inspector that affect the gameplay statistics.

Attack Range is set automatically from the root node and does not need to be changed.
Graphics and the attached AnimationPlayer affect how the attack animation behaves.
The toolkit uses units that do not have animations for attacking and thus an alternative was invented: Instead of animating the sprites the attack is portrayed as a sword that appears above the unit, slashes and disappears.
Having sprites with attack animations would require some changes in the Unit-Graphics script and then the attack Graphics can be left empty. If the desire is to remove it then the Attack.cs would have to be updated accordingly.

2.2.2 Ability

Abilities are added purely through script. To make a new one one should choose between Ability and TargetedAbility. Ability is an abstract class and requires the implementation of what is to happen on using the Ability (through button click or keyboard shortcut and in case of AIplayers upon use). There is a virtual function OnClickUI that is to be used only for HumanPlayers and it is used to modify the UI.
A name of the Ability must be specified in the Text property and if one sets the cooldown greater than default 0 the Ability will not let the player use it before this time expires.
TargetedAbility inherits Ability but after performing the OnClick effect it also requires a target to be chosen and has functions that handle what is to happen on target selection, cancellation and what is to happen when the target is reached. The range property specifies at what distance is the target considered reached.
It is very important to specify that every Ability must have an accompanying resource. These are follow the naming convention of having the Ability's name with the "Res" suffix.
AbilityRes works as an initializer for its Ability and gets displayed in the inspector once implemented and the project is rebuilt.

2.2.3 Selectable

One chooses from the scenes: Selectable, Unit or Building and creates a new inherited scene (Available from the Scene tab or by pressing Ctrl+Shift+N) If one has chosen pure Selectable then it is required to also make a new script that inherits from the Selectable class as it is an abstract class and attach it to the root node. Core statistics of the Selectable can now be set in the inspector on the root node.

- S Name is a unique name that the Selectable will use in the UI when selected. It should be a distinct name as it could be incorrectly mixed together when sorting player's selection.

- Export Abilities is an array of pairs. The first value is an integer that indicated the position on the grid in the player's UI and the second value is the ability resource.

- Vision Range distance in Tilemeters of the distance the unit can see. (used for Agro range)

Damageables also have Max Hp and units have speed in TilesPerSecond.
Child nodes have following uses:

- CollisionShape2D as the name implies describes what shape the Selectable has. This object must keep the same name but the Type can be anything that inherits CollisionShape2D. Should your Selectable be stationary and have rectangular shape it is recommended to use the TileCollisionShape2D. It has its size is in Tilemeters, the object will snap to a Tilemap if it is specified and it will allow Units and other objects using navigation path around it.
- UnitPortrait sets the Texture in the UI when the Selectable is selected.
- Graphics are scene on its own and it consists of the following:
  - Selected is a polygon that is shown when the Selectable is selected. It should be shaped as a circular base under the Selectable or as an outline around it.
  - Sprite2D is the texture or a spritesheet of the Selectable
  - AnimationPlayer is used to animate the Sprite2D.
  - Pathline used by Units to draw the path they take or by Buildings to show where the Rally point is.
  - Cross is currently a graphic that flashes over Damageables when they die. It removes the need to have death animations on the sprite sheet.
  - DamageableGraphics (a scene that inherits from Graphics) also has Healthbar. It needs to be properly placed above the Damageable.
- VisionArea is used for vision and is already set through the root node.
- Attacks is a node that holds the Selectable's attacks. Attack scenes should be added as its children
- Abilities is a node that holds the Selectable's abilities. Abilities are already set in the root node so this node should not be touched and nothing should be placed as its child.
- Unit also has NavAgent that handles navigation. It should not be touched.

2.2.4 Map

To make or edit the Tilemap of the level it is important to note that for unit pathing to work the pathable tiles use "Navigation Layer 0" and collision is added through "Physics Layer 0" The tileset in the demo game has this specified on the id 33.
If a different tileset is used it is necessary to edit the "Replacement tile" inspector tab for all the TileCollisionShape2Ds (collider used by Buildings). The replacement tile must have collision and must not be pathable otherwise units would pass through buildings.

2.2.5 Game level

Game level can be done by creating a new Node2D and attaching the Map.cs script. Under this node it is necessary to put the map described above named Ground with the Ground.cs script and all players that will be present in the level.

2.2.6 Menu

The menu works much the same as Game levels do it is a node that has Gamelevel without HumanPlayers as a child along with the UI elements for menu buttons. One does not need to modify Menu if they are adding a map. Maps are automatically loaded from godot-rts/scenes/Levels and are checked for the ".tscn" filename extension and if they are valid scenes. It does not check whether the scene is a valid Level though so it is recommended not to put any scenes that are not game levels into the folder otherwise it may result in unexpected behavior. However Menu is only used to switch scenes and can be completely omitted if the user is so inclined. Just set a game level as the initial scene to skip Menu entirely. Button descriptions:

- Campaign is used as a simple scene switch to a specified scene.
- Chapters displays all scenes in the Levels folder.
- Settings currently does nothing but is prepared for user settings.
- Exit Game quits the game

3 Demo game

The demo game is strongly integrated with the rest of the toolkit and most of its components can serve as a reference point or template for the user.

3.1 Development documentation

The demo consists of the following:

- Menu.tscn plays a FFA at the Background and is technically a game level with the menu overlay and no way to interact (all players in this level are AIplayers) it serves as an entry point for the game and has basic functionality to load Gamelevels
- Level1.tscn is the only level in the game. It contains one human player and one AI player.
- The knight, king and archer unit are either present or trainable in the Level1. They have different movement speeds, attack speeds and abilities.
- Barrack - a building capable of spawning knights and archers.
- Two players each owning one Barrack unit each and a handful of units.
  One player is human and his forces are the users
  The other player is a blank player with no agenda. There just to hold on to the Units.
  Players are hostile to each other (they belong to different teams) and their units will automatically attack each other if in vision range.
- Terrain in the level uses the Ashlands texture and is mostly pathable dirt and gravel with small lava lakes and patches of water that are unpathable. The left and right sides of the map are mostly split by a body of water and are only connected on a few spots by narrow dirt bridges in the middle of the map.

3.2 User documentation

Upon opening the game there are menu buttons and a FFA battle at the background. Only interactable objects are the menu buttons.

- Campaign starts the only level and Chapters gives a list of all available levels (Lists only one)
- Settings button does not do anything.
- Exit Game exits the game.

Upon starting the Game level one can see the UI on the top and bottom of the screen and a view onto the game map filling the majority of the screens center. The UI consists of the following:

- In the top left there is the Menu button pressing which will pause the game and display options out of which only the "Exit to menu" and "Exit Game" are implemented. Goals and Log buttons do nothing.
- In the top right is where Resources would be located. There are two with default icons and are unused at the moment.
- In the bottom left there is a minimap that shows the entire map zoomed out.
- To the right of it there is the UnitsSelected/UnitInfo section. If a single Selectable is selected the UnitInfo is displayed and would show the Selectable's statistics. Should multiple Selectables be selected then the UnitsSelected gets displayed showing the list of selected units. The player can click the bookmark on the top of this box to switch between these two views. UnitInfo displays the statistics of Selectable first in the selection (order is decided from various factors within code)
- Next section to the right is the unit portrait and it displays the usually the face of Selectables or other texture related to the primary selected Selectable.
- And to the right of the Portrait in the bottom right there is a grid that holds the Abilities the selected Selectable can perform. Clicking on these has effects such as building a building or training a unit.

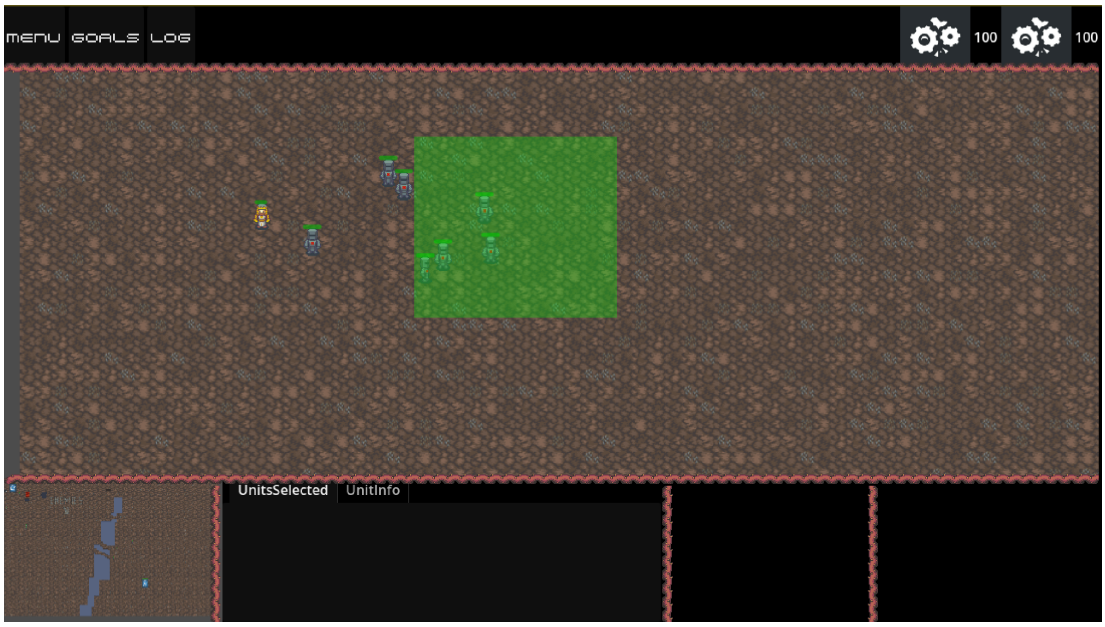

Figure 3.1 Selecting 4 knights

If the player presses and holds the left mouse button a rectangle will appear. Upon releasing the mouse all friendly Selectables will be selected. If no friendly Selectables are within the selection and there is at least one enemy Selectable it will be selected too but the player will not be able to issue commands to it - only read its statistics. Once one or more friendly units are selected right clicking on the map will make them move in that direction. If a player presses the A key

beforehand they will move to that location but will attack anything hostile on the way (if it enters their vision area) If the A key is pressed and the right click is on something that can be attacked for example unit (even friendly) or a building the selected Selectables will go attack it.

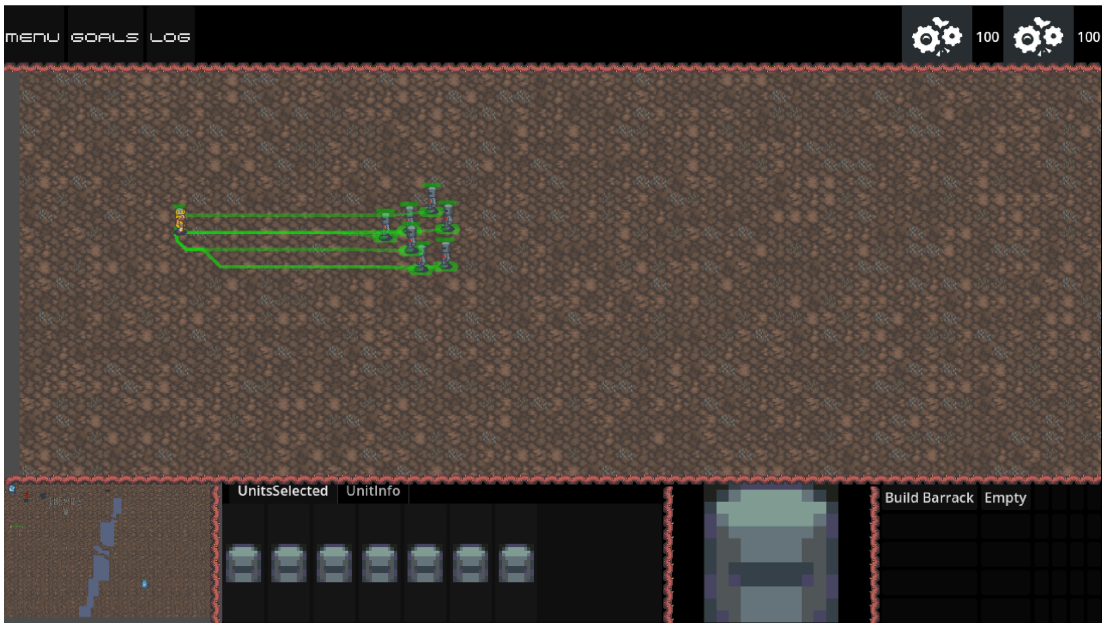

Figure 3.2 Units ordered to attack

Attacking can be seen by the red sword appearing and the health bars above the units decreasing. Once the bar is empty a cross appears across the target and it promptly disappears being dead.

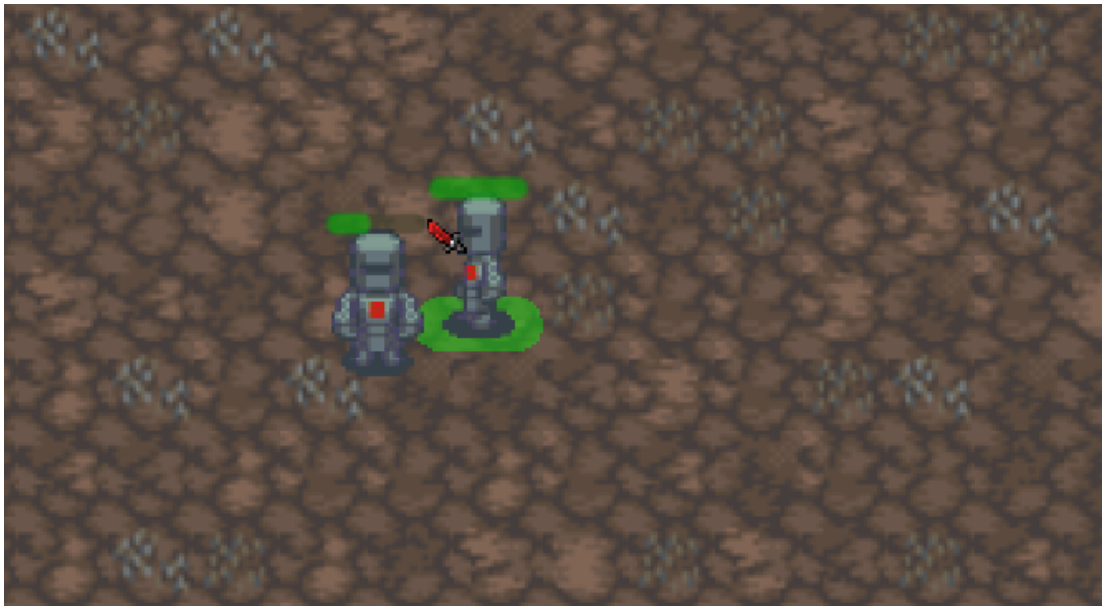

Figure 3.3 Attack animation

There are patches of water and lava on the ground that units cannot enter and will path around it if instructed to go past them.

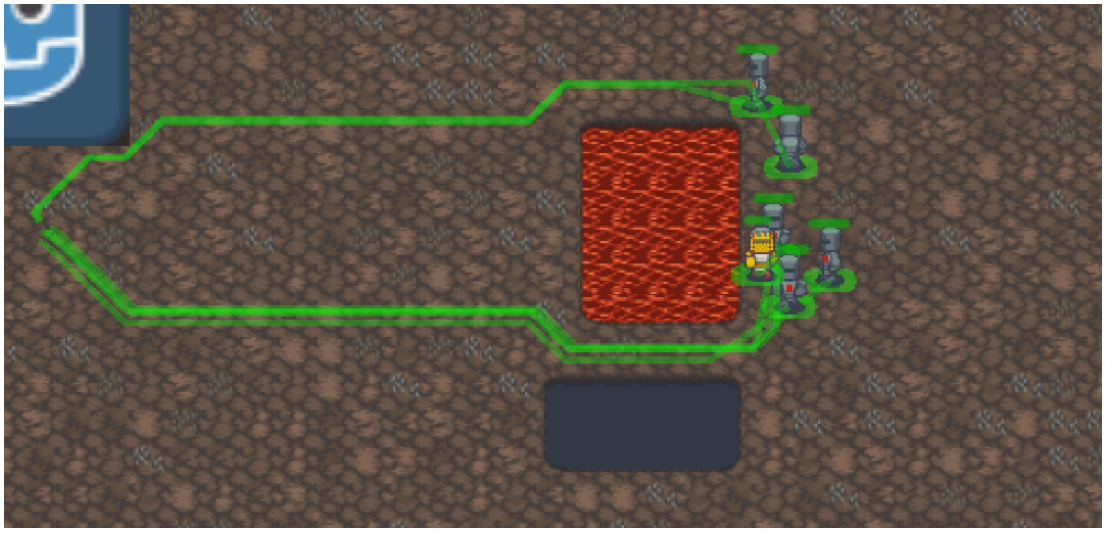

Figure 3.4 Units pathing around obstacles

By moving the mouse to the sides of the screen (initially just to the right and bottom) the view will shift in that direction revealing more of the map.
There are some enemies to the bottom right corner of the map with which can be fought. The player has one building at their disposal - the barracks using the placeholder Godot logo to present itself. This one can also be selected and has the ability to spawn more friendly units. There is a cooldown on spawning them which is about 1second long.
That is all that there is to do in the demo game.

4 Recommendations for future work

- Move core parts of the toolkit to C++ once there is a reliable way to generate C# glue. This will help separate the toolkit code from user code, integrate it better with the editor and increase performance which is always a an important factor for RTS games.
- Update alongside the engine to make use of new features and fixes, especially the C# integration.
- Write a competent AI that can make use of user-added content.
- Add multiplayer support.
- Improve UI modularity - resizable UI and other settings.
- Add support for scripted game content - custom campaigns and different game modes.
- Expand on Abilities - Area of Effect ones, Damage over time etc.

5 License

The toolkit is a free to use tool for the Godot engine and so the MIT license was chosen. Note that the graphical assets are not owned by the author of this thesis and follow their own licensing which is in all cases the Creative commons license.

```
 Copyright (c) 2024 Jiří Malý

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

6 Comparison with similar projects

6.1 Commercial RTS editors

Commercial RTS editors like the ones shipped with Warcraft 3 and StarCraft 2 have vastly more features, premade assets and maps, but are limited by the fact that their source code is not publicly available. Therefore the amount of features is fixed probably forever. They are also more or less stuck with the graphics of the game was made in. This toolkit is just an extension of Godot engine - if it can be done in Godot the toolkit can just help it be done faster giving the user all the power of modern and continually developed game engine. Godot also has reasonable backwards compatibility, often requiring minor changes when updating. (The toolkit begun on Godot 3.5 and has been updated to Godot 4.2 without much issue.)

6.2 ORTS engine

[41] ORTS engine's development stopped (last update to the engine appears to be 2007 and last update to their website in 2017). It was made mainly for AI researchers in mind to test AI's against each other but according to their website they hosted altogether 4 competitions with less than 10 submissions each time. In the author's opinion since the engine advertised itself too heavily on AI research and did not have a campaign or a player friendly game it was just obscure enough that most people didn't know it existed and it never got the traction to be interesting even for the researchers themselves. It is open source and written in C++. This toolkit is made more as a framework for hobbyist game developers who want to have a headstart when wanting to make a RTS in Godot. It is more user friendly than ORTS by using a better known engine and scripting language, but is much smaller project than ORTS as ORTS has fully fleshed networking and is an engine built from scratch rather then being an addon for existing one.

7 Conclusion

In conclusion the project is in a usable, functional state and can be used to create a very simple RTS game. The demo game is functional and uses no extra code than what is provided via the toolkit proving that it is possible to avoid scripting. However the demo is also very shallow gameplay-wise and shows that there is a need for further extension of the toolkit to make interesting games. It fulfills its primary purpose of providing a stepping stone for time constrained game developers to make prototypes and simple games, but it requires more work to be usable by people without programming skills.
As for the goals outlined in Section 1.5:

1. Conceptual goals

    - There is a very simple AI player that can use Selectables under his command to attack the opposing teams, but is not able to use Abilities and thus cannot train more units nor build more buildings.
    - The demo game proves that a simple game can be made without scripting.
    - The demo is playable and shows off the implemented mechanics
    - The demo game and the toolkit are preset to play similar to conventional RTS games - controls, camera, unit selection, abilities and UI layout are implemented as outlined in the Requirements section 1.4
    - The toolkit is open source and its classes are prepared to be extendable through inheritance and via components.

2. Technical goals

    - AI players, Selectables, Abilities and Attacks are all C# classes built with inheritance and composition in mind. There are abstract classes and virtual functions with code comments on how to implement them. All required class interface is public or protected to allow for inheritance without workarounds or frequent edits of the toolkit's code.

    Detailed documentation is provided for C# and C++ project and most critical functions and classes are annotated with code comments.
    It can be concluded that the goals have been fulfilled.

8 Acknowledgments

8.1 Inspirations

8.1.1 Bachelor thesis on RTS by Adam Hanka

HANKA, Adam. Engine for Real-time Strategy (RTS) Games. Bakalářská práce, vedoucí Ježek, Pavel. Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra distribuovaných a spolehlivých systémů, 2013.
The bachelor thesis of Adam Hanka has been used as a source for inspiration on the thesis structure and content due to the similarity of the subject matter, but at the time of reading it the toolkit was nearly finished and Hanka's code had not been used as a source of inspiration.

8.2 Books

The book "Godot Engine Game Development Projects: Build five cross-platform 2D and 3D games with Godot 3.0"[42] was very useful in the early stages when the project was still in Godot v3.5 and helped understand the basics of working with the Godot editor. However it focuses almost entirely on GDScript and is by now mostly outdated and is replaceable by the online documentation.

8.3 Use of AI in the thesis&project

AI tool ChatGPT3.5 [43] has been used sparingly during the writing of this thesis. No paragraphs or pieces of text have been directly copied from it. It has been used to help find a fitting word to a sentence or two to avoid unusual phrasing. It had been used to try debug some parts of the project's code but since GPT3.5 is stuck before January 2022 it does not have record of Godot4 and thus it was not much help and nothing ended up being used.

8.4 External Assets

All assets used in the project are under creative commons license and are credited in the folders they are stored in.

Bibliography

1. Wikipedia contributors. *Real-time strategy — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Real-time_strategy&oldid=1220092059. [Online; accessed 1-May-2024].
2. Blizzard Entertainment. *WarCraft III* [online]. 2002. [visited on 2024-05-01]. Available from: https://warcraft3.blizzard.com.
3. Blizzard Entertainment. *StarCraft* [online]. 1998. [visited on 2024-05-01]. Available from: https://starcraft.blizzard.com.
4. Blizzard Entertainment. *StarCraft 2* [online]. 2010. [visited on 2024-05-01]. Available from: https://starcraft2.blizzard.com.
5. Wikipedia contributors. *Multiplayer online battle arena — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Multiplayer_online_battle_arena&oldid=1213320444. [Online; accessed 1-May-2024].
6. Valve Corporation. *DOTA 2* [online]. 2013. [visited on 2024-05-01]. Available from: https://www.dota2.com/home.
7. Riot Games. *League of Legends* [online]. 2009. [visited on 2024-05-01]. Available from: https://www.leagueoflegends.com.
8. Buro, Michael. *ORTS - A Free Software RTS Game Engine* [online]. 2003. [visited on 2024-05-01]. Available from: https://skatgame.net/mburo/orts/.
9. Stratagus team. *Stratagus* [online]. [visited on 2024-05-01]. Available from: https://stratagus.com/.
10. Juan Linietsky, Ariel Manzur and contributors. *Godot Engine* [online]. 2007. [visited on 2024-05-01]. Available from: https://godotengine.org.
11. Wikipedia contributors. *Dune II — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Dune_II&oldid=1221295412. [Online; accessed 1-May-2024].
12. Wikipedia contributors. *Herzog Zwei — Wikipedia, The Free Encyclopedia*. 2023. Available also from: https://en.wikipedia.org/w/index.php?title=Herzog_Zwei&oldid=1174156406. [Online; accessed 1-May-2024].
13. Electronic Arts. *Command & Conquer* [online]. 1995. [visited on 2024-05-01]. Available from: https://www.ea.com/cs-cz/games/command-and-conquer.
14. Blizzard Entertainment. *World of Warcraft* [online]. 2004. [visited on 2024-05-01]. Available from: https://worldofwarcraft.blizzard.com/.
15. Wikipedia contributors. *Massively multiplayer online role-playing game — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Massively_multiplayer_online_role-playing_game&oldid=1220107338. [Online; accessed 1-May-2024].

16. Wikipedia contributors. *Defense of the Ancients — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Defense_of_the_Ancients&oldid=1218033375. [Online; accessed 1-May-2024].

17. Adams, Dan. The State of the RTS. *IGN*. 2012. Available also from: https://www.ign.com/articles/2006/04/08/the-state-of-the-rts.

18. Wikipedia contributors. *Real-time tactics — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Real-time_tactics&oldid=1220959331. [Online; accessed 2-May-2024].

19. Wikipedia contributors. *Company of Heroes — Wikipedia, The Free Encyclopedia*. 2023. Available also from: https://en.wikipedia.org/w/index.php?title=Company_of_Heroes&oldid=1190719492. [Online; accessed 2-May-2024].

20. Wikipedia contributors. *SpellForce — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=SpellForce&oldid=1213699991. [Online; accessed 2-May-2024].

21. Wikipedia contributors. *The Settlers — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=The_Settlers&oldid=1214641251. [Online; accessed 2-May-2024].

22. Wikipedia contributors. *City-building game — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=City-building_game&oldid=1221055498. [Online; accessed 2-May-2024].

23. Electronic Arts. *SimCity* [online]. 2024. [visited on 2024-05-01]. Available from: https://www.ea.com/games/simcity.

24. Creative Assembly. *TotalWar* [online]. 2024. [visited on 2024-05-01]. Available from: https://www.totalwar.com/.

25. Wikipedia contributors. *Role-playing video game — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Role-playing_video_game&oldid=1220047384. [Online; accessed 2-May-2024].

26. Wikipedia contributors. *Homeworld — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Homeworld&oldid=1215883284. [Online; accessed 5-May-2024].

27. Brazie, Alexander. *Best Game Engines for Beginner Game Developers in 2024* [online]. 2024. [visited on 2024-05-01]. Available from: https://gamedesignskills.com/game-development/video-game-engines/.

28. Epic Games. *Unreal Engine* [online]. 1998. [visited on 2024-05-01]. Available from: https://www.unrealengine.com/.

29. Unity Technologies. *Unity* [online]. 2005. [visited on 2024-05-01]. Available from: https://unity.com/.

30. Wikipedia contributors. *MIT License — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=MIT_License&oldid=1218118317. [Online; accessed 1-May-2024].

31. Godot contributors. *Godot Docs*. 2024. Available also from: https://godotengine.org/.

32. Raw Fury AB. *Dome keeper* [online]. 2022. [visited on 2024-05-01]. Available from: https://rawfury.com/games/dome-keeper/.

33. Chasing Carrots. *Halls of Torment* [online]. 2023. [visited on 2024-05-01]. Available from: https://godotengine.org/showcase/halls-of-torment/.

34. Mega Crit. *Slay the Spire 2* [online]. 2025. [visited on 2024-05-01]. Available from: https://store.steampowered.com/app/2868840/Slay_the_Spire_2/.

35. Mega Crit. *Slay the Spire* [online]. 2017. [visited on 2024-05-01]. Available from: https://store.steampowered.com/app/646570/Slay_the_Spire/.

36. Godot contributors. *Godot Changelog* [online]. 2024. [visited on 2024-05-01]. Available from: https://github.com/godotengine/godot/blob/master/CHANGELOG.md.

37. Godot contributors. *Godot Proposals* [online]. 2024. [visited on 2024-05-01]. Available from: https://github.com/godotengine/godot-proposals/issues/8191.

38. Brandon Blanker Lim-it. *Versus Series #1 - ECS vs OOP* [online]. 2020. [visited on 2024-05-01]. Available from: https://flamendless.github.io/ecs-vs-oop/.

39. Wikipedia contributors. *Programming paradigm — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Programming_paradigm&oldid=1219012176. [Online; accessed 6-May-2024].

40. Wikipedia contributors. *Multiple inheritance — Wikipedia, The Free Encyclopedia*. 2023. Available also from: https://en.wikipedia.org/w/index.php?title=Multiple_inheritance&oldid=1186885662. [Online; accessed 5-May-2024].

41. Buro, Michael; Furtak, Timothy. ON THE DEVELOPMENT OF A FREE RTS GAME ENGINE. In: 2005. Available also from: https://api.semanticscholar.org/CorpusID:14837497.

42. Bradfield, C. *Godot Engine Game Development Projects: Build five cross-platform 2D and 3D games with Godot 3.0*. Packt Publishing, 2018. isbn 9781788836425. Available also from: https://books.google.cz/books?id=KMNiDwAAQBAJ.

43. OpenAI. *ChatGPT* [online]. 2022. [visited on 2024-05-01]. Available from: https://chat.openai.com/.

List of Abbreviations

- RTS - Real-time strategy (game)
- OOP - object-oriented programming
- ECS - Entity component system
- MOBA - multiplayer online battle arena
- MMORPG - Massively multiplayer online role-playing game
- UI - User interface

A Attachments

A.1 Toolkit

Godot project in the godot-rts folder (installation guide in INSTALL.md)

A.2 Demo-game executable

Compiled for Windows x86_64.
For exporting to other platforms follow the guide in INSTALL.md