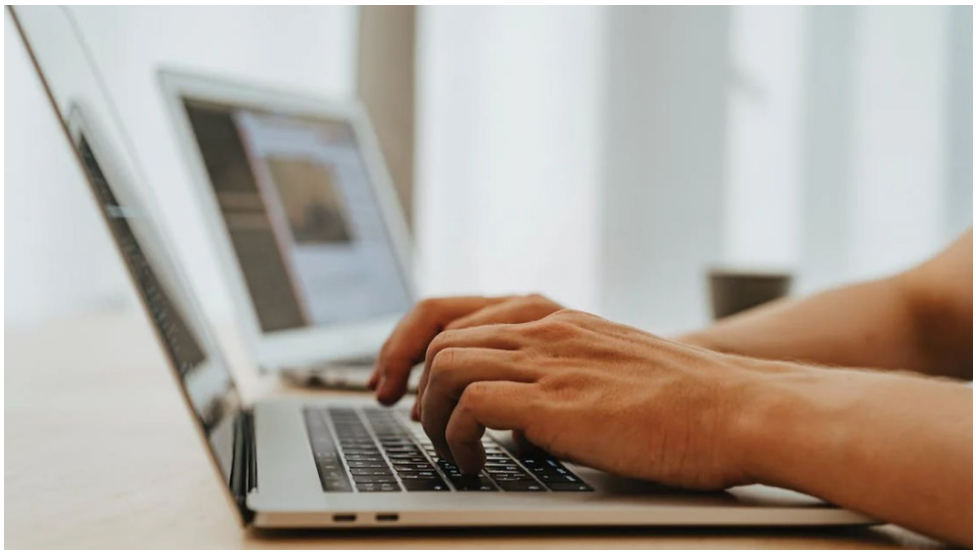
SŠPU STŘEDNÍ ŠKOLA PRŮMYSLOVÁ A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

Skirmish game



Autor: Walter Oboňa

Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

Třída: IT4

Školní rok: 2025/26

Prohlášení

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým a prezentačním účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 6. 1. 2026

. . . . . .

Podpis autora

Abstrakt

Tato závěrečná studijní práce se zabývá návrhem a implementací jednoduché tahové skirmish hry vytvořené v herním enginu Godot. Cílem práce bylo navrhnout funkční herní systém, který zahrnuje správu herních jednotek, tahový systém, pohyb po mapě, bojový mechanismus a uživatelské rozhraní.

Praktická část práce se zaměřuje na implementaci herní logiky pomocí skriptovacího jazyka GDScript, návrh herní mapy založené na dlaždicích a práci s herními objekty. Důraz je kladen na přehlednou strukturu kódu, využití objektově orientovaného přístupu a správu herních stavů.

Výsledkem práce je funkční prototyp tahové hry pro dva hráče, který demonstruje základní principy vývoje her a může sloužit jako základ pro další rozšíření.

Klíčová slova skirmish hra, tahová hra, Godot Engine, GDScript, herní logika, vývoj her

Abstract This final study project focuses on the design and implementation of a simple turn-based skirmish game developed using the Godot game engine. The main goal of the project was to create a functional game system including unit management, turn handling, tile-based movement, combat mechanics, and a basic user interface.

The practical part of the project describes the implementation of game logic using the GDScript programming language, the design of a tile-based game map, and the interaction between game objects. Emphasis is placed on clear code structure, the use of object-oriented programming principles, and proper management of game states.

The result of the project is a functional prototype of a two-player turn-based game that demonstrates fundamental principles of game development and can serve as a foundation for future extensions.

Keywords skirmish game, turn-based game, Godot Engine, GDScript, game logic, game development

Obsah

Úvod

Svět stolních a deskových her mě fascinoval odjakživa. Od rychlých karetních her až po komplexní strategie trvající několik hodin, které svými mechanikami představují výzvu i pro zkušené hráče. Při výběru tématu závěrečné práce proto byla moje volba jasná – chtěl jsem propojit svou vášeň pro deskové hry s programováním a přenést zážitek z fyzického stolu do digitálního prostředí.

Rozhodování o konkrétním titulu nebylo snadné. Zvažoval jsem adaptaci sběratelských karetních her nebo komplexních asymetrických strategií. Nakonec jsem však zvolil "zlatou střední cestu"– inspiraci hrou *Warhammer Warcry*. Jedná se o tzv. skirmishovou strategii, ve které se střetávají dvě menší skupiny jednotek. Cílem je eliminace soupeře.

Vývoj v herním enginu Godot se ukázal být náročnější, než jsem původně předpokládal, zejména kvůli nedostatku specifických materiálů pro tento typ her v novější verzi enginu. Proces byl provázen mnoha slepými uličkami a nutností opakovaně přepisovat části kódu (refactoring). Tato zkušenost však byla ve výsledku neocenitelná – naučila mě nejen lépe ovládat jazyk GDScript, ale především analyticky přemýšlet o architektuře softwarového projektu.

Předkládaná práce je rozdělena do čtyř hlavních částí. První kapitola představuje použité technologie. Druhá kapitola se věnuje popisu hry z pohledu hráče a adaptaci pravidel pro digitální prostředí. Třetí a čtvrtá kapitola pak detailně popisují technickou implementaci klíčových mechanik.

1 POUŽITÉ TECHNOLOGIE A NÁSTROJE

V této kapitole jsou představeny softwarové nástroje, které byly zvoleny pro realizaci projektu.

1.1 GODOT ENGINE

Pro vývoj hry byl zvolen herní engine Godot ve verzi 4.x. Jedná se o open-source nástroj (licence MIT), který poskytuje komplexní prostředí pro tvorbu 2D i 3D her. Oproti konkurenčním enginům vyniká svou lehkostí a unikátní architekturou.

1.1.1 Architektura uzlů a scén

Základním stavebním kamenem Godotu je systém uzlů (Nodes). Každý herní objekt, ať už jde o postavu, mapu nebo zvukový přehrávač, je uzel. Uzly se skládají do stromové struktury (Scene Tree). Tato hierarchie umožňuje vysokou modularitu.

1.1.2 Systém signálů

Pro komunikaci mezi objekty využívá Godot návrhový vzor Observer, implementovaný formou signálů. To umožňuje, aby objekty vysílaly zprávy o změně svého stavu (např. „jednotka dokončila pohyb“), aniž by musely znát příjemce této zprávy.

1.2 JAZYK GDSCRIPT

Logika hry je psána v jazyce GDScript, který byl vyvinut speciálně pro potřeby enginu Godot. Jedná se o vysokoúrovňový jazyk inspirovaný Pythonem. Mezi klíčové vlastnosti jazyka využité v této práci patří:

- **Volitelné statické typování:** Zvyšuje bezpečnost kódu a výkon aplikace.
- **Klíčové slovo @export:** Umožňuje vystavit proměnné ze skriptu přímo do grafického editoru (Inspektoru).

1.3 ASEPRITE

Pro tvorbu grafické stránky hry (pixel art) byl využit editor Aseprite. Jedná se o specializovaný nástroj pro tvorbu 2D spritů a animací, který je v herním průmyslu standardem pro retro grafiku.

1.4 ALGORITMUS A* (A-STAR)

Pro hledání cesty (pathfinding) v tahových strategiích je standardem algoritmus A*. V projektu je využita implementace `AStarGrid2D`, která je optimalizovaná přímo pro mřížkové mapy.

2 POPIS HRY

Tato kapitola popisuje vytvořenou hru z pohledu hráče a vysvětluje klíčová rozhodnutí učiněná během vývoje.

Je to tahová strategie typu *skirmish* (šarvátka) pro dva hráče. Hra se odehrává na čtvercové síti, která reprezentuje bojiště s různými typy terénu a překážkami. Každý hráč ovládá malou skupinu jednotek. Cílem hry je taktickou převahou eliminovat všechny jednotky soupeře. Hra končí v momentě, kdy jeden z hráčů přijde o svou poslední jednotku.

2.1 ADAPTACE PRAVIDEL A ODLIŠNOSTI OD PŘEDLOHY

Ačkoliv herní koncept vychází z pravidel figurkové hry *Warhammer Warcry*, přímá konverze fyzické hry do digitální podoby vyžadovala řadu designových změn a zjednodušení. Cílem nebylo vytvořit věrnou kopii 1:1, ale využít výhod, které digitální prostředí nabízí.

2.1.1 Přechod od spojitého prostoru k diskrétnímu

Nejvýraznější změnou je systém pohybu. Původní stolní předloha se odehrává ve spojitém prostoru, kde hráči měří vzdálenosti fyzickým metrem. Pro účely této práce byl prostor diskretizován do čtvercové mřížky (gridu). Toto rozhodnutí přináší několik výhod:

- **Jednoznačnost:** Odpadají spory o to, zda je figurka "ještě v dosahu".
- **Algoritmizace:** Mřížka umožňuje snadnou implementaci pathfinding algoritmu A* a výpočet vzdáleností pomocí Manhattanské metriky.
- **Přehlednost:** Hráč okamžitě vidí (díky vizuálnímu overlayi), kam až může jeho jednotka dojít.

2.1.2 Automatizace herních mechanismů

Zatímco ve stolní verzi musí hráči ručně házet kostkami, počítat výsledky a kontrolovat tabulky zranění, digitální verze tyto procesy automatizuje. Hra v každém kroku sama validuje pravidla

(např. Line of Sight pomocí Bresenhamova algoritmu) a okamžitě aplikuje výsledky soubojů. Tím se eliminuje lidská chyba a zrychluje se průběh herního kola.

2.2 PRŮBĚH HERNÍHO KOLA

Hra je rozdělena na kola (Rounds). Každé kolo se skládá ze dvou hlavních fází: fáze iniciativy a fáze akcí.

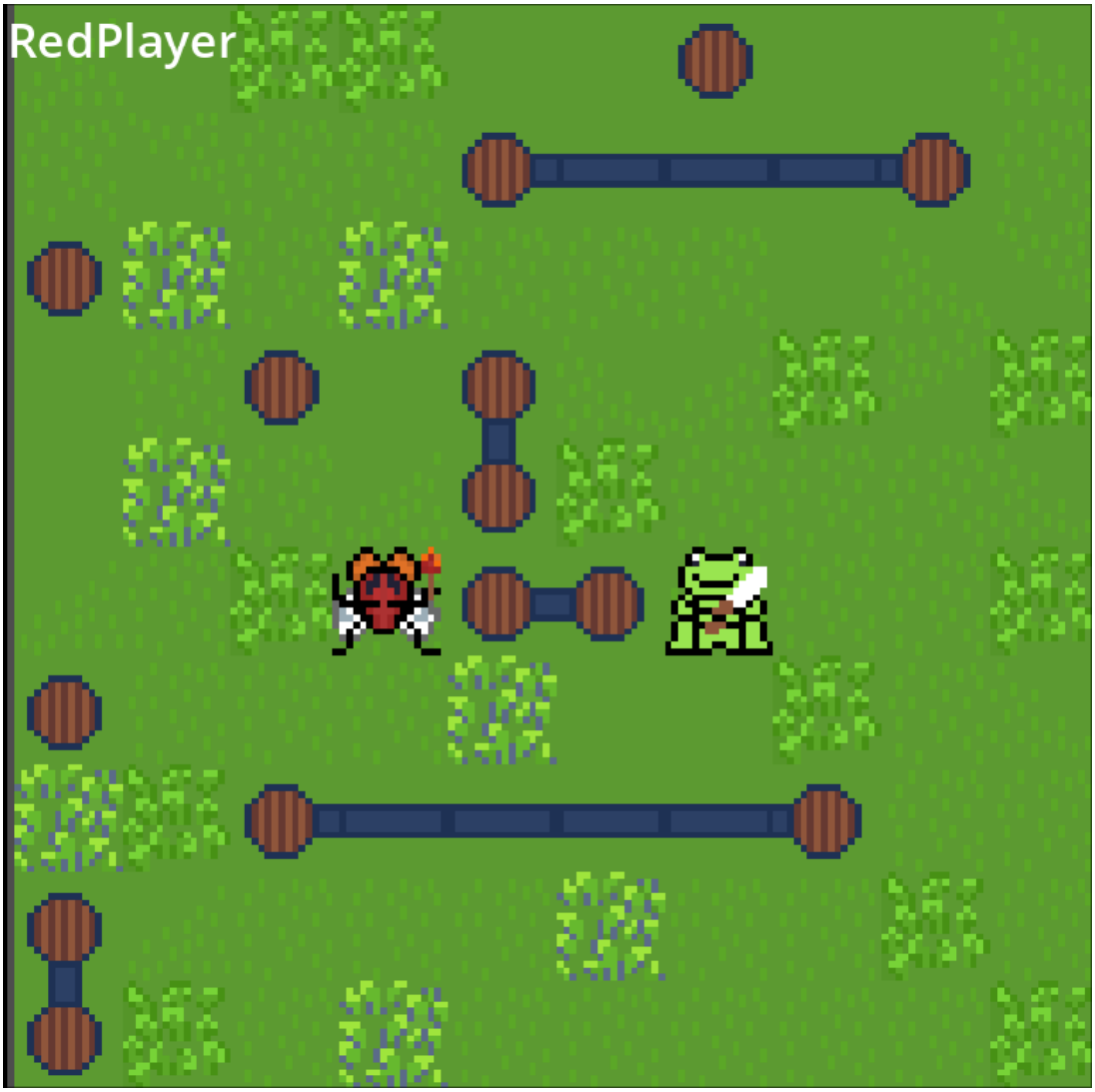
2.2.1 Fáze iniciativy

Na rozdíl od klasických her jako šachy, kde se hráči střídají fixně, využívá tento projekt prvek náhody pro určení pořadí. Na začátku kola proběhne „hod kostkami“ (vizuálně reprezentovaný v UI). Hráč, kterému padne nejméně stejných hodnot, získává právo prvního tahu.

2.2.2 Fáze akcí (Turn)

Jakmile je určeno pořadí, hráči se střídají v aktivaci svých jednotek. Hráč může s aktivní jednotkou provést následující operace:

1. **Pohyb:** Jednotka se přesune o počet polí definovaný její statistikou. Pohyb spotřebuje 1 akční bod.
2. **Útok:** Pokud je nepřítel v dosahu a je na něj vidět, jednotka může zaútočit. Útok spotřebuje 1 akční bod.
3. **Vyčkávání (Wait):** Hráč může tah ukončit předčasně. Tato akce spotřebuje akční bod a předá tah soupeři, ale jednotka zůstane aktivní pro budoucí kola.
4. **Disengage (Odpoutání se):** Pokud se jednotka nachází v bezprostřední blízkosti nepřítele (sousední políčko), je považována za vázanou v boji. Její maximální pohyb je pro tento tah penalizován (omezen na fixní vzdálenost 3 polí). Tato mechanika simuluje taktickou obtížnost bezpečného ústupu od protivníka.



Obrázek 2.1: Ukázka herní plochy s rozmístěnými jednotkami a překážkami.

3 NÁVRH A ARCHITEKTURA APLIKACE

Tato kapitola popisuje vnitřní strukturu projektu, organizaci herních objektů a vztahy mezi nimi. Důraz je kladen na modularitu systému a využití návrhových vzorů, které usnadňují budoucí rozšiřitelnost aplikace.

3.1 VOLBA VÝVOJOVÉHO PROSTŘEDÍ

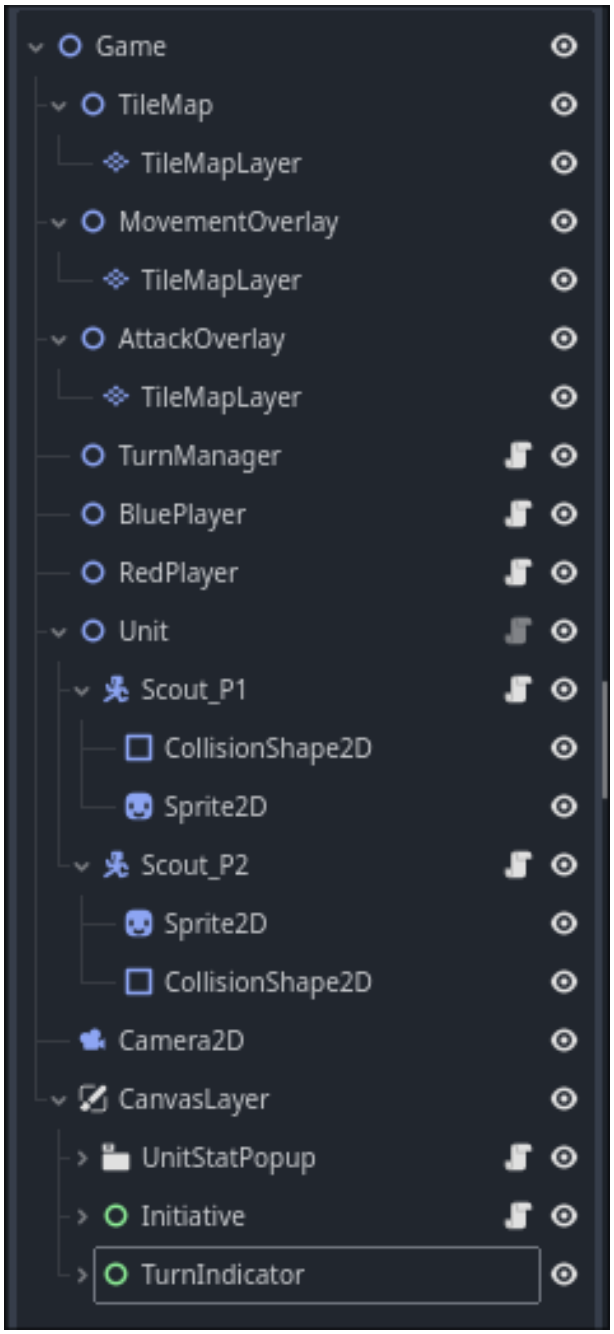
Pro realizaci projektu byl jako hlavní nástroj zvolen herní engine Godot (verze 4.x). K tomuto rozhodnutí vedlo několik klíčových faktorů, které tento engine zvýhodňují oproti konkurenčním řešením specificky pro potřeby 2D tahové strategie:

- **Dedikovaný 2D engine:** Na rozdíl od mnoha jiných enginů, které simulují 2D prostor v 3D světě, Godot pracuje s pravými 2D souřadnicemi. To výrazně zjednodušuje výpočty pohybu na mřížce a práci s pixel-art grafikou.
- **Architektura uzlů (Nodes):** Hierarchický systém scén v Godotu je ideální pro objektový návrh deskové hry. Každá jednotka, políčko mapy nebo prvek UI může být samostatný uzel s vlastním skriptem, což usnadňuje správu kódu.
- **Jazyk GDScript:** Vestavěný skriptovací jazyk umožňuje velmi rychlé prototypování herní logiky a algoritmů bez zbytečné složitosti kompilovaných jazyků.

3.2 HIERARCHIE HERNÍ SCÉNY

Základem projektu je hlavní scéna `Game.tscn`, která funguje jako kořenový uzel (Root Node). Organizační struktura využívá systém rodič-potomek, což je v enginu Godot klíčové pro správné fungování transformací (pozicování) a relativních cest ve skriptech.

Rozvržení uzlů přesně kopíruje logické vrstvy aplikace (od pozadí po popředí):



Obrázek 3.1: Hierarchie uzlů v hlavní herní scéně (Scene Tree).

1. **TileMap (Prostředí):** Nejnižší vrstva obsahující vizuální reprezentaci mapy a data o terénu (zdi, podlaha).

2. **MovementOverlay & AttackOverlay:** Dvě samostatné vrstvy pro vizualizaci herních mechanik. Slouží k vykreslování modrých (pohyb) a červených (útok) políček nad mapou.

3. **TurnManager:** Neviditelný řídící uzel. Neobsahuje grafiku, ale drží hlavní skript `turn_queue.gd`, který řídí pravidla hry.

4. **BluePlayer & RedPlayer:** Uzly typu `Player`, které fungují jako kontejnery. Každý z nich v sobě drží své podřízené jednotky (`Unit`). Tím je jasně definováno vlastnictví a zjednodušuje se procházení smyčkou "přes všechny moje jednotky".

5. **CanvasLayer (UI):** Vrstva uživatelského rozhraní, která je vykreslována nezávisle na kameře. Obsahuje tři hlavní pod-systémy:

   - **TurnIndicator:** Zobrazuje, kdo je na tahu.
   - **Initiative:** Řídí vizualizaci hodů kostkou na začátku kola.
   - **UnitStatPopup:** Informační okno se statistikami jednotky.

3.3 Architektura řízení hry (Core Loop)

Jádro herní logiky je postaveno na úzké spolupráci tří klíčových tříd, které tvoří páteř celého systému. Tuto strukturu lze označit jako centrální řídící trojúhelník:

1. **TurnManager (Orchestrátor):** Tato třída funguje jako mozek hry. Nedrží data o konkrétních jednotkách, ale spravuje stavy hry (čí je tah, fáze iniciativy, fáze akce). Rozhoduje, který hráč je na řadě, a předává mu řízení.

2. **Player (Správce zdrojů):** Představuje abstrakci hráče. Jeho úkolem není řešit pohyb (to dělá Unit) ani pravidla kola (to dělá TurnManager), ale spravovat své "zdroje"– tedy seznam svých živých jednotek a výsledky hodů na iniciativu. Funguje jako prostředník mezi globálním manažerem a konkrétními figurkami.

3. **Unit (Výkonná jednotka):** Třída, která reálně provádí akce v herním světě (pohyb, útok). Je to koncový bod architektury. Unit neví, čí je kolo, pouze ví, že byla aktivována a má vykonat příkaz.

3.3.1 Tok řízení (Control Flow)

Interakce mezi těmito komponentami je cyklická:

1. `TurnManager` vybere aktivního `Player`.
2. `Player` poskytne seznam dostupných `Unit`.
3. Hráč (uživatel) klikne na `Unit`, čímž ji aktivuje.
4. `Unit` vykoná akci (např. pohyb) a pomocí signálu oznámí `TurnManageru`, že akce skončila.
5. `TurnManager` vyhodnotí situaci a případně předá tah druhému hráči.

3.4 KOMUNIKACE MEZI KOMPONENTAMI

Aby byl kód modulární a jednotlivé části na sobě nebyly tvrdě závislé (tight coupling), využívá architektura návrhový vzor **Observer**, který je v Godotu implementován pomocí **signálů**.

Tento přístup umožňuje oddělit herní logiku od vizualizace:

- **Jednotka → Manažer:** Když jednotka dokončí pohyb, nevysílá příkaz "přepni tah", ale pouze emituje signál `movement_finished`. Je na manažerovi hry (`TurnManager`), jak s touto informací naloží.
- **Jednotka → UI:** Po kliknutí na jednotku je emitován signál `unit_selected`. UI manažer tento signál zachytí a aktualizuje informační panel, aniž by jednotka musela vědět o existenci nějakého panelu.

3.5 SPRÁVA VLASTNICTVÍ A HIERARCHIE HRÁČŮ

Hra implementuje striktní hierarchii vlastnictví pro rozlišení, které jednotky patří kterému hráči. Ve scéně existují dva hlavní uzly typu `Player` (např. *RedPlayer* a *BluePlayer*). Každý z nich funguje jako kontejner:

- **Správa jednotek:** Jednotky jsou přímými potomky uzlu hráče. Při startu hry skript `player.gd` projde své potomky a zaregistruje je do pole `units`.
- **Dice Pool:** Každý hráč si drží svá data o hodech na iniciativu.

Toto uspořádání zjednodušuje logiku střídání tahů – manažer hry pouze přepne aktivního hráče a automaticky tím získá přístup k jeho sadě jednotek.

3.6 OBJEKTOVÝ NÁVRH A DĚDIČNOST

Pro zajištění snadné rozšiřitelnosti (např. přidání nových typů postav) je využita dědičnost.

3.6.1 Abstraktní třída Unit

Základním stavebním kamenem je třída `Unit`. Tato třída definuje společné chování pro všechny entity na bojišti:

- Pohyb po křivce (Tweening).
- Detekci kliknutí (InputEvent).
- Správu akčních bodů.

3.6.2 Specializace

Konkrétní typy jednotek, jako je například Průzkumník (`Scout`), dědí ze základní třídy. Díky dědičnosti není nutné v nových jednotkách znovu programovat pohyb či boj. Potomek pouze v metodě `_ready()` přepíše výchozí hodnoty statistik (viz Datový model v kapitole 4) a případně upraví vizuální stránku.

4 IMPLEMENTACE HERNÍCH MECHANIK

Tato kapitola analyzuje klíčové technické výzvy, které vyvstaly během vývoje, a popisuje zvolená algoritmická řešení.

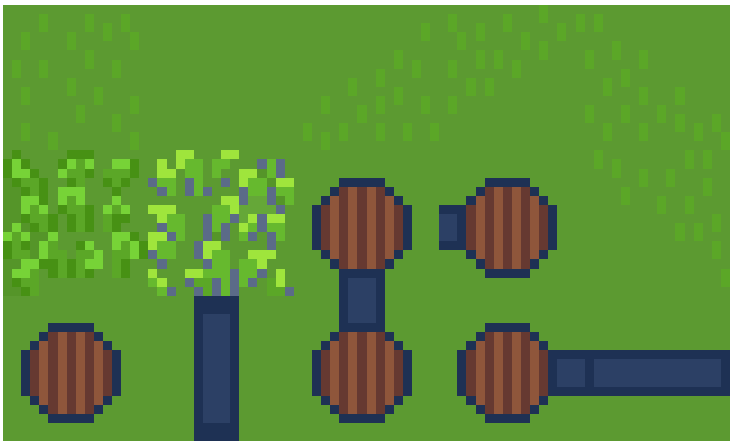
4.1 GRAFICKÉ PODKLADY (ASSETS)

Před samotným programováním logiky bylo nutné připravit grafické podklady. Hra využívá styl *pixel art* s rozlišením mřížky 16x16 pixelů, což je standard pro retro 2D hry.

4.1.1 TileSet (Sada dlaždic)

Pro vykreslení herního světa není použit jeden velký obrázek celé mapy, což by bylo paměťově náročné, ale tzv. **TileSet**. Jedná se o jeden texturový atlas (spritesheet), který obsahuje všechny varianty terénu (tráva, hlína, kamenné zdi, voda) naskládané vedle seba.

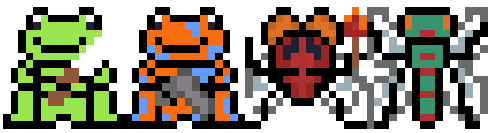
Godot Engine tento obrázek virtuálně rozřeže na čtverce o velikosti 16x16 bodů. Každému políčku je přiděleno unikátní ID souřadnice v atlasu. Díky tomu je mapa v paměti uložena pouze jako pole čísel (referencí na ID dlaždice), nikoliv jako obrovská bitmapa.



Obrázek 4.1: Použitý TileSet (atlas). Z jednoho obrázku jsou poskládány všechny variace prostředí. Všimněte si organizace do mřížky.

4.1.2 Sprity jednotek

Zatímco prostředí je statické, postavy jsou dynamické objekty. Grafika jednotek je řešena pomocí samostatných textur (Spritů). Na rozdíl od TileSetu, který je "přilepen"k mřížce, uzel `Sprite2D` umožňuje plynulý pohyb po sub-pixelech (viz interpolace pohybu v sekci **??**). Každá jednotka (např. Scout) má svou vizuální reprezentaci, která je vycentrována (offset) tak, aby nohy postavy stály přesně uprostřed herního políčka.



Obrázek 4.2: Detail spritů herních jednotek.

4.2 Návrh mapy a datové vrstvy

Základem herního prostředí je mřížková mapa vytvořená pomocí systému `TileMapLayer`. V Godotu 4 neslouží tento systém pouze k vykreslování grafiky, ale funguje i jako nositel důležitých herních dat.

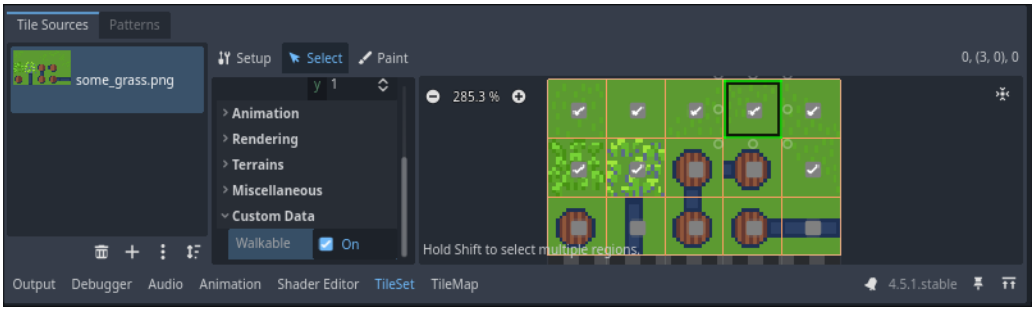
4.2.1 Konfigurace TileSetu

Pro mapu byl vytvořen vlastní `TileSet`. Aby bylo možné algoritmicky rozlišit, která políčka jsou průchozí (podlaha) a která tvoří překážku (zdi, propasti), byla využita funkce **Custom Data Layers**.

Byla definována booleovská vrstva s názvem `Walkable`.

- **True (Průchozí):** Přiřazeno dlaždicím podlahy a trávy.
- **False (Neprůchozí):** Přiřazeno dlaždicím zdí, stromů a vody.

Toto nastavení umožňuje oddělit vizuální stránku od logické. Programátor nemusí řešit, jak konkrétní zeď vypadá, zajímá ho pouze datový atribut na daných souřadnicích.



Obrázek 4.3: Editor TileSetu v Godotu. V pravém panelu je vidět nastavení vlastnosti "Walkable"pro vybranou dlaždici zdi (hodnota false).

4.2.2 Přístup k datům v kódu

Díky tomuto přístupu je kontrola průchodnosti v GDScriptu velmi efektivní. Není potřeba počítat fyzikální kolize. Místo toho se kód dotáže přímo mapových dat na konkrétní souřadnici mřížky.

Níže uvedená ukázka demonstruje, jak třída `TurnManager` získává informace o terénu:

```
# Získání datové buňky na souřadnicích (x, y)
var tile_data = tile_map.get_cell_tile_data(Vector2i(x, y))

# Pokud buňka existuje a má nastaveno Walkable na true, je průchozí
if tile_data and tile_data.get_custom_data("Walkable"):
    return true
else:
    return false
```

Kód 4.1: Čtení Custom Data z mapy

Tento systém je využit jak při generování navigační sítě pro A* (viz sekce 4.2), tak při kontrole viditelnosti pro střelbu (Bresenhamův algoritmus).

4.3 Datový model a atributy jednotek

Před popisem samotných herních mechanismů je nutné definovat datovou strukturu herních objektů. Každá jednotka ve hře je instancí třídy, která nese sadu parametrů ovlivňujících její chování v boji i při pohybu. Tyto atributy jsou definovány v základní třídě `Unit` a upravovány v jejích potomcích (např. třída `Scout`).

Klíčové atributy zahrnují:

- **Health Points (HP):** Představují celkovou vitalitu jednotky. Při snížení hodnoty na 0 je jednotka vyhodnocena jako zničená a následně odstraněna z herní scény.
- **Movement Points:** Maximální vzdálenost (v počtu polí), kterou může jednotka urazit během jednoho tahu.
- **Attack:** Kvantifikuje objem útoku. Číselná hodnota odpovídá počtu kostek, kterými se hází při vyhodnocení souboje.
- **Far:** Definuje maximální efektivní dostřel jednotky (vyjádřeno v počtu políček mřížky).
- **Strength (Síla) & Toughness (Odolnost):** Dvojice protichůdných atributů (ofenzivní vs. defenzivní). Jejich vzájemným porovnáním se určuje obtížnost hodu na zásah.
- **Hit & Crit:** Konstanty definující výši způsobeného poškození. *Hit* se aplikuje při standardním zásahu, zatímco *Crit* se aplikuje při padnutí hodnoty 6 na hrací kostce.

4.4 Navigace v herním prostoru

Jedním ze základních požadavků tahové strategie je schopnost jednotek inteligentně se pohybovat v prostředí plném překážek. Bylo nutné implementovat systém, který nejen najde cestu z bodu A do bodu B, ale také zohlední terén a pohybový limit jednotky.

4.4.1 Pathfinding (Algoritmus A*)

Pro řešení problému hledání nejkratší cesty v grafu byl implementován algoritmus A* (A-Star). Volba padla na tento algoritmus pro jeho efektivitu a schopnost využití heuristické funkce k optimalizaci vyhledávání. V rámci Godot Enginu byla využita třída `AStarGrid2D`, která je

optimalizována přímo pro mřížkové mapy. Tím odpadla nutnost manuální konstrukce grafu sousednosti. Pro metriku vzdálenosti byla zvolena **Manhattanská vzdálenost** (pohyb pouze ortogonálně), což odpovídá pravidlům deskové předlohy.

4.4.2 Vizuální interpolace pohybu

Okamžitá změna souřadnic jednotky (teleportace) by působila nepřirozeně a ztěžovala by hráči sledování průběhu tahu. Pro zajištění plynulé vizualizace byla nasazena technika interpolace (Tweening). Místo skokové změny pozice se využívá objekt `Tween`, který v čase interpoluje globální pozici spritu mezi jednotlivými body vypočítané cesty.

```
func move_along_path(path: Array[Vector2]) -> void:
    # ... (ošetření vstupů)
    var tween = create_tween()

    for point in path:
        # Lineární interpolace s vyhlazením pro přirozený dojem
        tween.tween_property(self, "global_position", point, 0.25)\
            .set_trans(Tween.TRANS_SINE)\
            .set_ease(Tween.EASE_IN_OUT)

    tween.tween_callback(Callable(self, "_on_move_finished"))
```

Kód 4.2: Interpolace pohybu pomocí Tween

Výsledkem je plynulá animace přesunu, díky které hráč neztrácí orientaci v prostoru a může snadno vizuálně sledovat trasu, kterou jednotka urazila.

4.5 Bojový systém a detekce viditelnosti

Implementace soubojů vyžadovala vyřešení dvou problémů: jak určit validitu útoku (zda útočník vidí na cíl) a jak spravedlivě vyhodnotit výsledek střetu na základě statistik definovaných v datovém modelu.

4.5.1 Detekce viditelnosti (Line of Sight)

Pouhé ověření vzdálenosti k cíli není pro validaci útoku dostačující, neboť herní mapa obsahuje neprůchozí překážky (zdi). Jelikož hra nepracuje s fyzikálním enginem a kolizními tvary pro zdi, nebylo možné použít standardní `RayCast2D`. Jako řešení byl zvolen **Bresenhamův algoritmus** pro rasterizaci úsečky. Tento algoritmus umožňuje matematicky simulovat přímku

v diskrétní mřížce a efektivně kontrolovat, zda některá z buněk na spojnici mezi útočníkem a obráncem neobsahuje příznak *non-walkable*.

```
func has_line_of_sight_tile(start: Vector2i, end: Vector2i) -> bool:
    var dx = abs(end.x - start.x)
    var dy = -abs(end.y - start.y)
    var err = dx + dy
    var e2

    while true:
        # Kontrola datové vrstvy mapy na přítomnost překážky
        var data = tile_map.get_cell_tile_data(start)
        if data and not data.get_custom_data("Walkable"):
            return false

        if start == end: break

        # Výpočet dalšího bodu na mřížce
        e2 = 2 * err
        if e2 >= dy:
            err += dy
            start.x += sx
        if e2 <= dx:
            err += dx
            start.y += sy
    return true
```

Kód 4.3: Kontrola viditelnosti (Bresenham)

Implementace tohoto algoritmu zajišťuje férovost hry – hráč nemůže zasáhnout cíle skryté za zdí či překážkou, což odpovídá intuitivnímu očekávání chování ve fyzickém prostoru.

4.5.2 Algoritmus vyhodnocení útoku

Samotný průběh boje je determinován porovnáním statistik útočníka a obránce. Systém nejprve určí prahovou hodnotu pro úspěšný zásah (Target Number) na základě vztahu mezi silou útočníka (*Strength*) a odolností obránce (*Toughness*).

Platí následující pravidla pro určení hodnoty potřebné k zásahu:

- **Strength > Toughness:** Útočník má výhodu, pro zásah stačí hodnota 3+.
- **Strength = Toughness:** Síly jsou vyrovnané, standardní hodnota pro zásah je 4+.
- **Strength < Toughness:** Obránce má převahu, útočník musí hodit 5+.

Následně proběhne simulace hodu *N* šestistěnnými kostkami, kde *N* odpovídá atributu *Attack* útočníka. Každá kostka se vyhodnocuje samostatně:

1. **Neúspěch:** Hodnota kostky je nižší než stanovená prahová hodnota. Žádné poškození se neaplikuje.
2. **Zásah (Hit):** Hodnota je rovna nebo vyšší než práh, ale nižší než 6. Je uděleno poškození ve výši atributu *Hit*.
3. **Kritický zásah (Crit):** Na kostce padla hodnota 6. Místo standardního poškození se aplikuje hodnota *Crit*, která reprezentuje silnější úder.

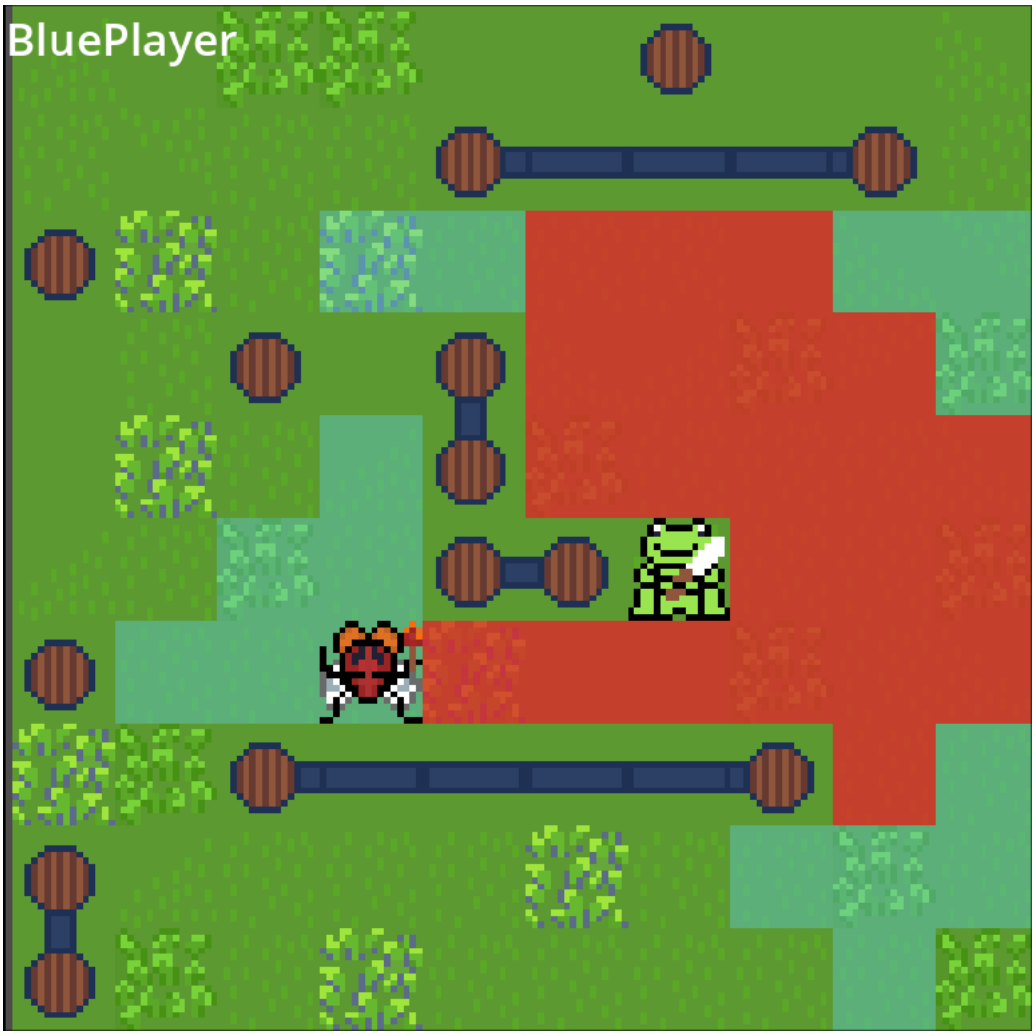
Celkové poškození je součtem výsledků všech úspěšných hodů. Tento mechanismus vnáší do hry prvek napětí a umožňuje strategické plánování rizika.

4.6 Vizualizace herních stavů

Zatímco architektura komunikace stojí na signálech (viz kapitola 3.2), samotná vizualizace pro hráče je řešena přímou manipulací s grafickými uzly.

4.6.1 Vykreslování dosahu (Overlays)

Pro zobrazení kam se jednotka může pohnout, není efektivní instancovat stovky samostatných objektů. Místo toho je využita metoda `set_cells_terrain_connect` na uzlu `TileMapLayer`. Po výběru jednotky se vypočítá pole dostupných souřadnic a tyto buňky jsou v jedné operaci přebarveny v dedikované vrstvě `MovementOverlay`.



Obrázek 4.4: Vizualizace taktických vrstev. Modré podbarvení (MovementOverlay) indikuje dostupný pohyb pro aktuální tah, zatímco červené zvýraznění (AttackOverlay) vymezuje oblast v dosahu útoku, kde je možné zasáhnout nepřítele.

4.6.2 Animace UI prvků

Pro plynulý zážitek (tzv. "game feel") nejsou informační okna zobrazována skokově. Při zobrazení detailu jednotky (`popupmenu.gd`) je využita interpolace vlastnosti `modulate:a` (průhlednost) a `scale`, což vytváří efekt plynulého objevení (fade-in).

ZÁVĚR

Cílem této práce bylo navrhnout a vytvořit funkční tahovou strategii typu skirmish v herním enginu Godot, která čerpá inspiraci z pravidel stolní hry Warhammer Warcry. Záměrem bylo adaptovat většinu herních mechanik fyzické předlohy pro digitální prostředí. Mohu konstatovat, že tento cíl byl splněn a výsledná aplikace obsahuje kompletní hratelné jádro hry.

Výsledná aplikace demonstruje komplexní herní systém, který v sobě propojuje logiku deskových her s možnostmi moderního enginu. Podařilo se implementovat robustní systém pohybu po mřížce využívající algoritmus A* v kombinaci s plynulou vizualizací (Tweening). Plně funkční je rovněž bojový systém, který automatizuje porovnávání statistik jednotek a vyhodnocování hodů kostkou. Hra správně řeší střídání tahů včetně fáze iniciativy a pomocí systému Line of Sight (Bresenhamův algoritmus) zajišťuje férovou detekci viditelnosti přes překážky.

Vývoj v prostředí Godot Engine pro mě představoval velkou výzvu. Vzhledem k tomu, že jsem s tímto nástrojem začínal a na internetu chyběly ucelené návody pro tvorbu specificky tohoto žánru (tahová skirmish strategie), probíhala realizace převážně formou samostudia a experimentování. Tento proces učení se za pochodu (tzv. "learning by doing") s sebou přirozeně nesl nutnost častého refactoringu a přepisování již hotových částí kódu, když jsem přišel na vhodnější způsob implementace herních mechanik. V důsledku toho lze v projektu nalézt pasáže, kde jsou podobné problémy řešeny odlišnými přístupy. Tato nekonzistence je odrazem postupného vývoje mých znalostí a hledání optimálních řešení přímo v průběhu tvorby aplikace.

Projekt, ačkoliv funkční, otevírá široký prostor pro další rozvoj a obohacení herních mechanik. Do budoucna bych se rád zaměřil na zvýšení znovuhratelnosti vytvořením **širší palety herních map** a implementací **rozmanitějších úkolů** (objectives), aby každá partie nabízela odlišnou strategickou výzvu. Z technického hlediska by hru výrazně posunulo zavedení **výškových úrovní terénu** (elevation), které by do taktiky vneslo nový vertikální rozměr.

Dalším logickým krokem je rozšíření herního obsahu. Současný model dvou týmů by bylo vhodné nahradit **komplexním systémem frakcí**, kde by každá strana disponovala unikátními jednotkami a speciálními schopnostmi (abilities). S tím souvisí i plánovaná mechanika **tvorby vlastní čety** (squad customization) na základě bodového systému, která by hráčům umožnila strategicky skládat tým před bitvou. Vrcholem budoucího vývoje by pak mohly být nástroje pro hráče, umožňující **vytváření vlastních frakcí a jednotek**.

Tvorba této práce pro mě byla velkým osobním i odborným přínosem. Nejenže jsem si osvojil práci v moderním herním enginu a prohloubil znalost programování v jazyce GDScript, ale především jsem se naučil přemýšlet nad architekturou většího softwarového projektu. Celý proces vývoje mě navíc velmi bavil, neboť mi umožnil v praxi propojit mou dlouhodobou vášeň pro stolní hry s programováním. Schopnost rozdělit komplexní problém na menší, řešitelné celky a dotáhnout vlastní vizi do podoby funkčního produktu považuji za nejcennější zkušenost, kterou jsem během psaní práce získal.

https://github.com/Dvojak/Skirmish-game

LITERATURA

[1] *Godot Engine 4.x Documentation* [online]. Godot Engine Project, 2024 [cit. 2026-01-05]. Dostupné z: https://docs.godotengine.org

[2] *GDScript reference* [online]. Godot Engine Project, 2024 [cit. 2026-01-05]. Dostupné z: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html

[3] *A* Search Algorithm* [online]. GeeksforGeeks, 2024 [cit. 2026-01-05]. Dostupné z: https://www.geeksforgeeks.org/a-search-algorithm/

[4] *Bresenham's line algorithm* [online]. Wikipedia, 2024 [cit. 2026-01-05]. Dostupné z: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm