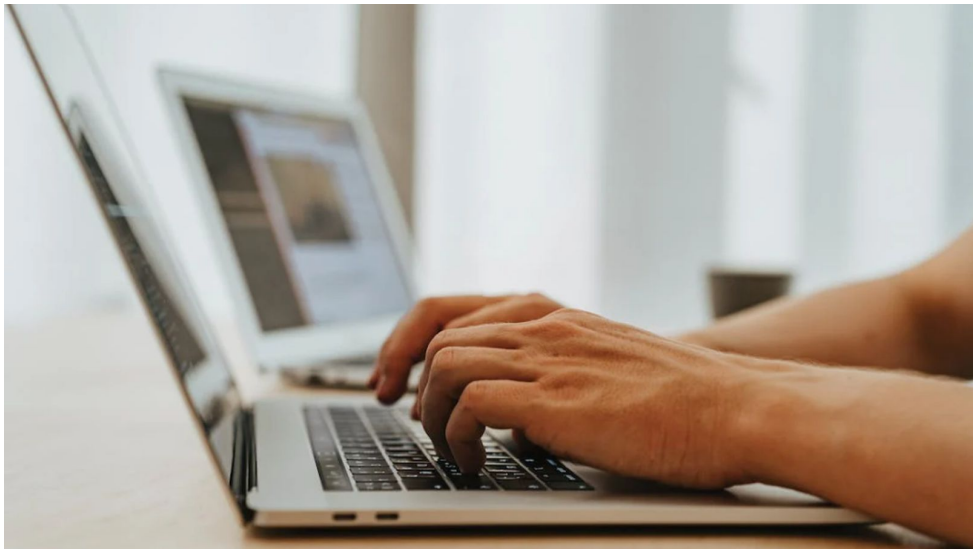
superiorSup superiorSup

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, jako je střídání hráčů a vyhodnocování soubojů.

Výsledkem práce je funkční prototyp tahové hry pro dva hráče, který demonstruje základní principy vývoje her a může sloužit jako základ pro další rozšíření, například o umělou inteligenci, nové jednotky nebo komplexnější herní mechaniky.



## 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 such as turn switching and combat resolution.

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, such as artificial intelligence, additional units, or more advanced game mechanics.



## 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, jako je *Magic: The Gathering*, kterými jsem se inspiroval již v minulosti, nebo naopak komplexních asymetrických strategií typu *Root*. Nakonec jsem však zvolil "zlatou střední cestu"– hru *Warhammer Warcry*.

Jedná se o tzv. skirmishovou strategii, ve které se střetávají dvě menší skupiny jednotek. Cílem je buď eliminace soupeře, nebo splnění taktických úkolů. Tato hra nabízí ideální rovnováhu: její pravidla jsou srozumitelná, ale zároveň poskytují dostatečnou hloubku pro implementaci zajímavých programátorských výzev, jako je pohyb po mřížce (gridu) nebo systém viditelnosti ve 2D prostoru.

Samotný vývoj v herním enginu Godot se ukázal být náročnější, než jsem původně předpokládal. Záhy jsem narazil na nedostatek specifických výukových materiálů a tutoriálů pro tento konkrétní typ hry, což mě donutilo hledat vlastní řešení a experimentovat. 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, zejména engine Godot a grafický editor Aseprite. Druhá kapitola se věnuje návrhu architektury aplikace a struktuře herní scény. Třetí, nejobsáhlejší kapitola, detailně popisuje technickou implementaci klíčových mechanik, jako je algoritmus A* pro hledání cesty, Bresenhamův algoritmus pro viditelnost nebo systém iniciativy. Závěr práce pak shrnuje dosažené výsledky a hodnotí funkčnost vytvořeného prototypu.

# 1 POUŽITÉ TECHNOLOGIE A NÁSTROJE

V této kapitole jsou představeny softwarové nástroje, které byly zvoleny pro realizaci projektu, a teoretické principy klíčových algoritmů.

## 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, jako jsou Unity nebo Unreal Engine, vyniká Godot 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 – komplexní objekty (např. hráč) jsou složeny z jednodušších uzlů (Sprite, Kolizní tvar, Kamera), které lze uložit jako samostatnou scénu a instancovat ji v jiných částech projektu.

### 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. Tento přístup zajišťuje oddělení logiky (decoupling) a přehlednější kód.

## 1.2 JAZYK GDSCRIPT

Logika hry je psána v jazyce GDScript, který byl vyvinut speciálně pro potřeby enginu Godot. Jedná se o vysokoúrovňový jazyk, jehož syntaxe je silně inspirována jazykem Python (využívá odsazování bloků kódu).

Mezi klíčové vlastnosti jazyka využité v této práci patří:

- **Volitelné statické typování:** Ačkoliv je GDScript dynamický jazyk, verze 4.x klade důraz na definici typů proměnných (např. `var health: int`). To zvyšuje bezpečnost kódu, umožňuje lepší našeptávání v editoru a zvyšuje výkon aplikace.
- **Klíčové slovo @export:** Umožňuje vystavit proměnné ze skriptu přímo do grafického editoru (Inspektoru). Designér tak může upravovat parametry hry (síla útoku, rychlost pohybu) bez nutnosti zasahovat do zdrojového kódu.
- **Integrace s C++ API:** GDScript má přímý přístup k nízkoúrovňovým funkcím enginu, jako jsou matematické operace s vektory nebo fyzikální výpočty.

## 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. Mezi klíčové funkce využité v tomto projektu patří:

- Práce s vrstvami (Layers) pro oddělení obrysů a barev.
- Tvorba bezešvých textur (Seamless Tiles) pro herní mapu.
- Export do formátu Sprite Sheet, který Godot Engine efektivně zpracovává při vykreslování animací.

## 1.4 Algoritmus A* (A-Star)

Pro hledání cesty (pathfinding) v tahových strategiích je standardem algoritmus A*. Tento grafový algoritmus hledá nejkratší cestu mezi startem a cílem pomocí heuristické funkce, která odhaduje zbývající vzdálenost. V projektu je využita implementace `AStarGrid2D`, která je optimalizovaná přímo pro mřížkové mapy.

# 2 NÁVRH A ARCHITEKTURA APLIKACE

Tato kapitola popisuje vnitřní strukturu projektu, organizaci herních objektů a vztahy mezi nimi.

## 2.1 STRUKTURA HERNÍ SCÉNY

Základem projektu je hlavní scéna `Game.tscn`, která funguje jako kořenový uzel a sdružuje herní logiku, mapu a uživatelské rozhraní. Organizační struktura využívá systém rodič-potomek, což je klíčové pro správné fungování transformací a relativních cest ve skriptech.

Níže uvedený seznam zobrazuje hierarchii uzlů v projektu (v závorce je uveden typ nebo připojený skript):

- **Game** (Root Node)
  - **TileMap** – *Vykreslování herního světa*
  - **MovementOverlay** – *Vrstva pro zobrazení dosahu pohybu*
  - **AttackOverlay** – *Vrstva pro zobrazení dosahu útoku*
  - **TurnManager** (`turn_queue.gd`) – *Správce tahů a pravidel hry*
  - **Players** (Kontejnery pro hráče)
    - player1 (`player.gd`)
    - player2 (`player.gd`)
  - **Unit** (Kontejner pro jednotky ve scéně)
    - Scout_P1 (`scout.gd`)
    - Scout_P2 (`scout.gd`)
  - **CanvasLayer** (Uživatelské rozhraní)
    - UnitStatPopup (`popupmenu.gd`) – *Informační okno jednotky*
    - Initiative (`initiative.gd`) – *Panel iniciativy*

## 2.2 OBJEKTOVÝ NÁVRH A DĚDIČNOST

Architektura kódu je postavena na principu dědičnosti, což umožňuje snadné rozšiřování hry o nové typy jednotek bez nutnosti přepisovat základní logiku.

### 2.2.1 Třída Unit

Základní třída `Unit` (soubor `unit.gd`) definuje společné vlastnosti všech postav. Obsahuje:

- **Atributy:** Zdraví (`health_points`), pohyb (`movement_points`), útok (`attack`).
- **Signály:** `movement_finished`, `no_actions_left` pro komunikaci s manažerem hry.

### 2.2.2 Specializace - Třída Scout

Konkrétní jednotky, jako je Průzkumník (soubor `scout.gd`), dědí ze třídy `Unit`. V metodě `_ready()` přepisují základní statistiky:

```
extends Unit
class_name Scout

func _ready():
  # Nastavení statistik specifických pro Scouta
  movement_points = 6
  health_points = 8
  toughness = 3
  type = "Scout"
```

Kód 2.1: Ukázka dědičnosti ve třídě Scout

# 3 IMPLEMENTACE HERNÍCH MECHANIK

Tato kapitola detailně popisuje programová řešení klíčových prvků hry. Zaměřuje se na algoritmy pro správu tahů, výpočet pohybu v mřížce a matematický model soubojového systému včetně řešení viditelnosti (Line of Sight).

## 3.1 ŘÍZENÍ HERNÍ SMYČKY (TURN SYSTEM)

Jádrem herní logiky je skript `turn_queue.gd`, který funguje jako centrální stavový automat. Hra se dělí na kola, která se skládají z tahů jednotlivých hráčů.

### 3.1.1 Iniciativa

Pořadí hráčů není fixní, ale určuje se na začátku každého kola pomocí systému iniciativy. Každý hráč disponuje sadou virtuálních kostek (reprezentovaných polem `dice_pool` ve třídě `Player`).

Metoda `roll_initiative_dice()` vygeneruje pro každou kostku náhodnou hodnotu v rozsahu 1–6. Hráč s nejnižším počtem stejných hodnot získává právo prvního tahu. Tento prvek náhody nutí hráče adaptovat strategii na každé nové kolo.

## 3.2 POHYB A PATHFINDING

Pohyb jednotek po herní ploše je realizován pomocí grafového algoritmu A* (A-Star), který hledá nejkratší cestu mezi startovní a cílovou dlaždicí s ohledem na překážky.

### 3.2.1 Konfigurace navigační mřížky

V projektu je využit uzel `AStarGrid2D`, který je optimalizován pro mřížkové mapy. Při inicializaci (`_ready`) se graf synchronizuje s vrstvou `TileMap`:

```
astar_grid = AStarGrid2D.new()
astar_grid.region = tile_map.get_used_rect()
astar_grid.cell_size = Vector2(16, 16)
# Zákaz diagonálního pohybu pro zachování čtvercové metriky
astar_grid.diagonal_mode = AStarGrid2D.DIAGONAL_MODE_NEVER
astar_grid.update()
```

Kód 3.1: Inicializace A* mřížky v turn_queue.gd

Byla zvolena **Manhattanská metrika** (pohyb pouze ve směru os X a Y), což zjednodušuje počítání vzdálenosti pro hráče – jeden krok vždy stojí jeden akční bod.

### 3.2.2 Interpolace pohybu (Tweening)

Samotná změna pozice jednotky není ve hře okamžitá. Aby byl pohyb vizuálně plynulý, využívá se objekt `Tween`, který provádí lineární interpolaci souřadnic v čase.

Metoda `move_along_path` ve třídě `Unit` přijímá pole bodů cesty a postupně mezi nimi animuje sprite jednotky:

```
func move_along_path(path: Array[Vector2]) -> void:
  if path.is_empty(): return

  actions -= 1
  var tween = create_tween()

  for point in path:
    # Animace přesunu na další bod cesty trvá 0.25 sekundy
    tween.tween_property(self, "global_position", point, 0.25)\
      .set_trans(Tween.TRANS_SINE)\
      .set_ease(Tween.EASE_IN_OUT)

  # Callback po dokončení pohybu
  tween.tween_callback(Callable(self, "_on_move_finished"))
```

Kód 3.2: Plynulý pohyb pomocí Tween (unit.gd)

## 3.3 BOJOVÝ SYSTÉM

Boj je interakcí mezi útočníkem a obráncem, která podléhá pravidlům o dosahu, viditelnosti a statistikách jednotek.

### 3.3.1 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:

- Pokud `Strength` $>$ `Toughness`: Útočník má výhodu, pro zásah stačí hodnota **3+**.
- Pokud `Strength` $=$ `Toughness`: Síly jsou vyrovnané, standardní hodnota pro zásah je **4+**.
- Pokud `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.

### 3.3.2 Atributy jednotek

Každá jednotka je definována sadou parametrů, které ovlivňují výsledek boje. Tyto atributy jsou definovány ve třídě `Unit` a upravovány v potomcích (např. `Scout`).

- **Health Points (HP):** Představují celkovou vitalitu jednotky. Při snížení hodnoty na 0 je jednotka vyhodnocena jako zničená a následně odstraněna z herní scény.

- **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.

### 3.3.3 Detekce viditelnosti (Line of Sight)

Aby mohla jednotka zaútočit na dálku, musí mít na cíl přímou viditelnost. Protože hra nefunguje na fyzikálním enginu, ale na mřížce, nemohl být použit standardní `RayCast2D`.

Místo toho byla implementována vlastní variace **Bresenhamova algoritmu**. Tento algoritmus matematicky simuluje přímku mezi dvěma body v mřížce a kontroluje, zda některá z buněk na této přímce neobsahuje překážku (dle vlastnosti *Custom Data: Walkable* v TileSetu).

Metoda `has_line_of_sight_tile` prochází buňky od útočníka k cíli. Pokud narazí na zeď, vrátí `false` a útok je znemožněn.

```gdscript
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

  while true:
    # Pokud je na aktuální dlaždici překážka, není vidět
    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 kroku v mřížce
    var e2 = 2 * err
    if e2 >= dy:
      err += dy
      start.x += sx
    if e2 <= dx:
      err += dx
      start.y += sy
  return true
```

Kód 3.3: Algoritmus kontroly viditelnosti (zjednodušeno)

## 3.4 Vizualizace a uživatelské rozhraní (UI)

Komunikace hry s hráčem probíhá ve dvou rovinách: přímo v herním prostoru (zvýraznění možností pohybu) a v statické vrstvě rozhraní (HUD).

### 3.4.1 Interaktivní vrstvy (Overlays)

Pro vizualizaci taktických možností, jako je dosah pohybu nebo útoku, hra nevyužívá drahé instancování objektů pro každé políčko. Místo toho jsou použity dedikované vrstvy typu `TileMapLayer`:

- **MovementOverlay:** Zobrazuje modře políčka, na která může jednotka dojít.
- **AttackOverlay:** Zobrazuje červeně políčka, která jsou v dosahu útoku.

Tyto vrstvy jsou ve stromu scény umístěny nad základní mapou, ale pod jednotkami. Při výběru jednotky skript `turn_queue.gd` vypočítá validní souřadnice (pomocí A* pro pohyb nebo Manhattanské vzdálenosti pro útok) a na příslušné souřadnice v overlay vrstvě umístí dlaždici s poloprůhlednou texturou:

```
func draw_movement_overlay(unit, reachable_points):
  overlay_map.clear() # Smazání předchozího stavu
  for point in reachable_points:
    # Nastavení modré dlaždice na souřadnici
    overlay_map.set_cell(point, SOURCE_ID, ATLAS_COORDS)
```

Kód 3.4: Vykreslení overlaye pohybu

Toto řešení je vysoce výkonné, protože `TileMapLayer` vykresluje všechny dlaždice v jednom draw-callu (vykreslovacím cyklu).

### 3.4.2 Statické rozhraní (HUD)

Informační prvky, které se nemají pohybovat s kamerou (např. statistiky), jsou umístěny ve vrstvě `CanvasLayer`.

Klíčovým prvkem je skript `popupmenu.gd`, který reaguje na signál `unit_selected`. Po kliknutí na jednotku se dynamicky načtou její aktuální statistiky do připravených textových polí (`Label`) a zobrazí se informační okno.

# ZÁVĚR

Cílem této závěrečné práce bylo navrhnout a naimplementovat tahovou strategickou hru v herním enginu Godot. Tento cíl byl splněn. Výsledná aplikace obsahuje funkční systém střídání tahů, pohyb jednotek po mřížce pomocí algoritmu A* a komplexní soubojový systém založený na statistikách jednotek.

Během vývoje jsem si prohloubil znalosti objektově orientovaného programování v jazyce GDScript a naučil se řešit specifické problémy herního vývoje, jako je oddělení herní logiky od vizualizace nebo práce s diskrétní geometrií na mřížce.

Vytvořený projekt představuje solidní základ, na kterém lze v budoucnu stavět, ať už přidáním umělé inteligence, nebo rozšířením herního obsahu. Práce prokázala, že Godot Engine je mocným nástrojem pro tvorbu 2D her, který umožňuje efektivní vývoj i pro jednotlivce.

# 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`


# Seznam obrázků

# Seznam tabulek