**Wandering Woods Game:**

Project Documentation

Aaron Matayka

&  
Andrew Matayka

# Table of Contents

[**Table of Contents 2**](#_criohazeerum)

[**1. Introduction 3**](#_za57m5n6dwoa)

[**a. Purpose 3**](#_68uic0wn7rfd)

[**b. The game 3**](#_wxedipygps3j)

[**2. Process Model 4**](#_m03q48smwm5a)

[**3. Use cases 5**](#_iphz3hnwptni)

[**a. K-2 5**](#_w3ugan74gjue)

[**b. 3-5 5**](#_tprv7kqnqgwe)

[**c. 6-8 5**](#_yqsgg236l5fh)

[**4. UML Model 7**](#_au987wsbflk4)

[**a. Use Case Diagram 7**](#_6rl0msy3ikz6)

[**b. Class Diagram 7**](#_otjih8uj990q)

[**c. State Diagram 7**](#_p0rv8wy2iug3)

[**5. Software Requirements Specification 8**](#_haeqz340luu2)

[**a. Functional Requirements 8**](#_lxdl5wshg6vc)

[**b. Non-functional Requirements 8**](#_obpktracaul)

[**c. External Interfaces 8**](#_w2yr7nacixbm)

[**6. Software Design 9**](#_bp816mkzpj4q)

[**a. Overall architecture 9**](#_exsihkhpbfy3)

[**b. Component Breakdown 9**](#_w6thnu15ypyn)

[**c. Algorithm and Data Structures 9**](#_njli2l4m8wzp)

[**d. Error handling 9**](#_z5tjns9l3bw)

[**7. Implementation 9**](#_u20sc79pjlq2)

[**8. User’s Guide 9**](#_o92gpq2ynz3s)

# Introduction

## Purpose

This document describes the software design of the Wandering Woods Game Project. This document is provided to clarify the design and function of the project and how it is to be used.

This project was developed in an agile method, with a focus on delivering the most important features for the client first, and focusing on other features after. This document will hereby only include information that is vital to the operation of the software and key functions.

## The game

The Wandering woods game is a simple game designed to be used by multiple grades as children grow up, as a demonstration of the advancement of knowledge and processing acquired through the years of schooling. There are three different variants of the program, which can be accessed from a main menu: K-2, 3-5 and 6-8. The K-2 variant is the simplest, with only a square 4x4 grid and 2 players to simply watch the program. The 3-5 variant allows the user to change things to be more advanced, offering a rectangle grid, and the ability to place players wherever they want. It also offers more information on the results of the game, showing more statistics overall. Finally, the 6-8 variant allows changing the movement type used in the game, further increasing the variance users can do. This allows the most freedom in program control overall. The game is overall a simple simulation of players being lost in the woods, and having to take “moves” to find each other. They cannot hear or see each other, and have to move without the knowledge of the other people’s locations. Upon 2 people finding each other, if relevant, they will combine into one group and search for the other people together. Upon finding all people, the game will end with a happy animation and display statistics on the run, such as speed to find each other, move amounts for each player, etc. Happy music will also play in the background for the k-2 game.

# Process Model

For our project, we weren’t sure what process model we would use, but ended up using the evolutionary process model in the end. This allowed us to keep making rapid prototypes of what we needed and keep making new prototypes over time. This is reflected in our commits as our general structure and python files changed drastically with each commit. This allowed us to work very quickly at the project and spin out new revisions at a higher speed than other models would offer.

# Use cases

## K-2

**Primary Actor:** K-2 grade child.

**Preconditions:**  A 4x4 grid is created with 2 players in opposite corners of the grid for viewing.

**Description:** A simple program that has 2 players in it in a 2x2 box, in which they always move randomly. The game plays with happy music in the background until they collide, then it stops and displays some basic statistics.

## 3-5

**Primary Actor:** 3-5 grade year child

**Preconditions:**  A grid of any user size is created with 2+ players in any positions of the grid for viewing.

**Description:** An intermediate program that has 2+ players in it in a player-defined grid, in which they always move randomly. The game plays until all players collide, then it stops and displays some more advanced statistics. When 2 players collide but there are more than two players, they group and find the other players together.

## 6-8

**Primary Actor:** 6-8 grade year child

**Preconditions:**  A grid of any user size is created with 2+ players in any positions of the grid for viewing, with the ability to change the roaming type as well from random.

**Description:** An advanced program that has 2+ players in it in a player-defined grid, in which they can move in multiple ways. The game plays until all players collide, then it stops and displays some more advanced statistics. When 2 players collide but there are more than two players, they group and find the other players together. Can track information on multiple runs and how they compare.

# 

# UML Model

## Use Case Diagram

The three use cases we have are shown in Figure 1 below, with the user representing the child using the program. Each of the three programs branch off to their own respective operations, with each calling the simulation for operation.

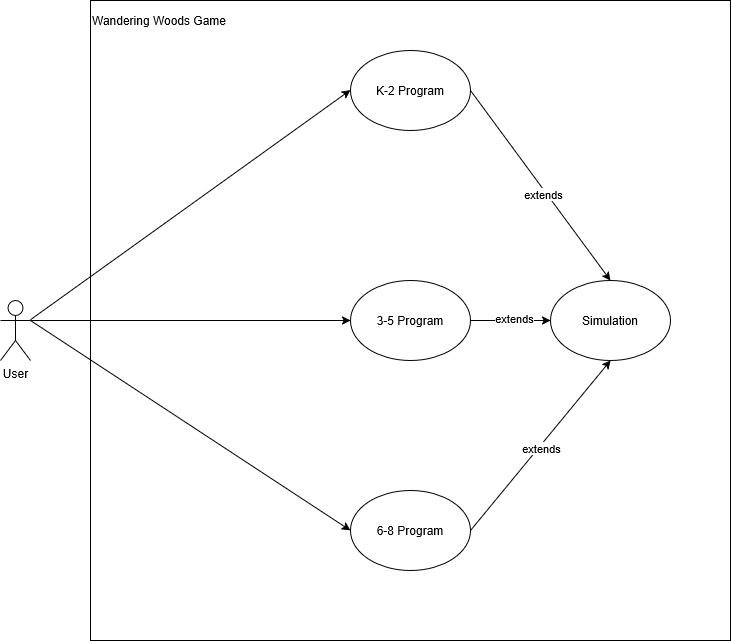


Figure 1

## State Diagram

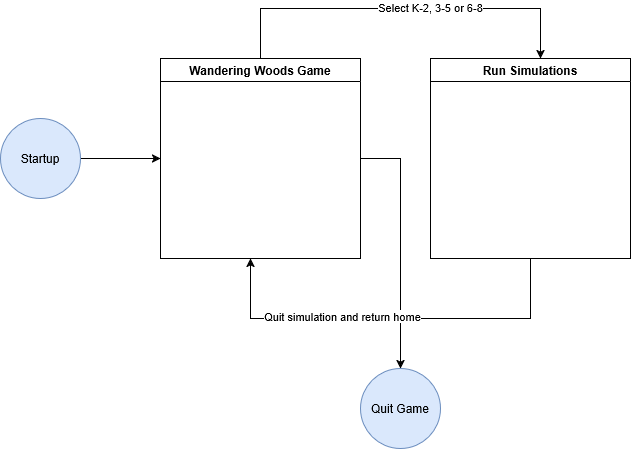
The state diagram in figure 2 shows high-level behavior, with the major states operated in for the game operation. It starts at startup where the user will pick the parameters. Then it is in the game state, where all options for running are picked. Then based on options picked we enter the simulations state, where it runs the simulation with the required stipulations. Then the program can go back to the main menu, and then quit with the quit game state. 

Figure 2

# Software Requirements Specification

## Functional Requirements

* + 1. Grid Generation and customization.
    2. Player Placement and movement.
    3. Game rules such as winning conditions and statistic tracking.
    4. User interaction like input and restart functionality.
    5. Audio for background music.
    6. Effect for winning in K-2

## Non-functional Requirements

* + 1. Performance concerns, ensuring it runs smoothly on old or slow systems.
    2. Usability, making interaction easy for students.
    3. Portability, it needs to be a simple executable.

## External Interfaces

* + 1. User gui, that can be clicked
    2. Can also use arrow keys for movement.
    3. Display is entirely in executable with a grid and multi-color circles for players

# Software Design

## Overall architecture

* + 1. See previous UML diagrams, simple feed from main code into 3 different sub-codes for different simulations

## Component Breakdown

* + 1. Game Grid: Creates and displays the grid of the game
    2. Player object: Stores internal functions for players, like movement, collision handling etc.
    3. Simulation base class: Core loop with different optional features for each type, also includes basic functions such as reset.
    4. Game Modes: Different variations for k-2, 3-5 and 6-8.
    5. Audio Handling: Imports a .wav file for k-2 happy music in the background.

## Algorithm and Data Structures

* + 1. Players are stored in lists for easy use, and the grid is stored as a 2d array of cells.
    2. Calculations are done with basic arithmetic for finding longest run, shortest run etc through counting and basic addition and subtraction.

## Error handling

* + 1. Uses simple try error exception handling for catching errors.
    2. Warn user if missing audio file.

# Implementation

* + 1. Used python for program implementation at all steps.
    2. Used pygame for music, ui, graphics etc.
    3. Test by bouncing code off each other and working to break each other’s code, in ways the other hadn’t thought of.

# User’s Guide

* + 1. To install, first go to the github repository located at <https://github.com/AaronMatayka/WanderingWoodsGame>
    2. Next, install the executable under “releases”, and run it on your system.
    3. Finally, launch the executable and play the game!