Team R.O.B

Documentation

Table of Contents

Members....3
- Kaden Gilbert....3
- Nova Solarz....3
- Gavin Sloan....3
- Nick Clark....3
- Jerry Cai....3

End-of-Project Summary....4
- Resulting Artifact....4

Preliminary Elements....5
- Executive Summary....5
- Outcome Summary....5
- Problem Statement....5
- Statement of Work....5
  - Project Objectives....5
  - Project Scope....5
  - Team Expectations....6
- Coding Standards and Practices Statement....6
- rough idea....6
- graphics....6
- sound design....6
- level design....7
- Requirements Security Requirements....7
  - Input Validation:....7
  - Unit Tests....7
  - GUI....7
  - OOP Practices....7
- User Requirements....7
- Hardware Requirements....8

Security....8
- Security Design Document....8
  - Assets to Consider....8
  - Threats to Assets....8
  - Security Measures....8
  - Security Summary....9
- Security Statement....9

Maintenance....9
- Maintenance Overview....9
- Maintenance Timeline....10
  - Week 1 of Development....10
  - Week 2 of Development....10

Week 3 of Development....10
Post-Release....10
External Asset Credits....11
Diagrams....12
Data Flow Diagram - L1....12
Problem Frame....13
UML....14

「 This section left intentionally blank 」

Members

Here listed are all of the members of Team R.O.B, along with the roles and duties each of us have taken on.

Note: Though we each have a specified role, we often perform duties that are not entirely contained within that role, as we are a small team and we all do our best to help each other.

Kaden Gilbert

(he / him)

- Team Lead
- Duties:
    - Unify team vision
    - Maintain fair division of labor

Nova Solarz

(it / they / she)

- Documentarian
- Duties:
    - Organize physical documentation
    - Make final passes of each article of documentation before submission to ensure quality and accuracy

Gavin Sloan

(he / him)

- Code Lead
- Duties:
    - Maintain and develop code-base
    - Collaborate with Design lead for structuring of our program

Nick Clark

(he / him)

- Design lead
- Duties:
    - Create diagrams to outline the functionality of our code artifacts
    - Collaborate with Code lead for structuring of our program

Jerry Cai

(he / him)

- Security Lead
- Duties:
    - Locate and notify Code lead about insecure coding practices
    - Hunt for bugs / parts of our program that can be improved

End-of-Project Summary

This summary outlines what we were able to achieve in the three-week time-frame that we were allowed in this SDLC exercise.

Resulting Artifact

Our final program artifact ended up being much different from what we had initially envisioned.

We did not:

- make the bulk of the visual assets,
- make any music or sound effects,
- implement a complex combat system,
- implement procedurally generated levels in conjunction with manually created levels,
- implement animations, or
- integrate a story for the player to follow into gameplay.

Though we did not complete everything we had intended to accomplish, we did create something of a functional game.

In our final artifact, we have:

- pixel-art visuals,
- background music,
- a primitive health/damage system (when the player collides with an enemy, both the player and the enemy attack at the same time, trading damage),
- multiple levels,
- multiple varieties of foes,
- a final boss, and
- win/lose states.

The game ended up becoming quite difficult as a result of its limitations, but it has been verified that it is possible to achieve the win-state.
Our Standards and Practices were largely dis-regarded in the final stages of our development due to time constraints and the crumbling mental health of our developers, i.e. we have a mashup of OOP and procedural programming, in addtion to code that I would relate to spaghetti, though saying that that would be an insult to all pasta-kind.

Preliminary Elements

Executive Summary

The problem we are trying to solve is eliminating boredom, stress, and restlessness by utilizing entertainment. The process we are taking to resolve this issue is developing a role-play video game with its design based off of the original Legend of Zelda game. We will be creating a main menu, various map features, background music, and other elements common in retro two-dimensional role-play games. Our proposed solution is one that will accommodate players of all ages who wish to relax after a long day, on a weekend, or when visiting friends and family. Our primary expected deliverable is our role-play game that will be gradually developed throughout this project.

Outcome Summary

Our expected outcomes are as follows:

- Our final product will be a fully functional two-dimensional role-play game.
- Implementing at least three separate "levels" in the game.
- Having a final boss by the end of the project's initial development.
- Having enemies located throughout the game's map.
- Having background music that plays during gameplay.
- Having a main menu that displays upon the game's startup.

Problem Statement

Our project aims to solve problems related to boredom, stress, and/or a lack of entertainment for the average person. By resolving these issues, we hope to provide a fun role-play style video game where the user can spend their free time exploring various "levels", items, and ultimately defeating a final boss.

Statement of Work

Project Objectives

- Create a fully functional two-dimensional role-play game
- Include a graphical user interface (GUI)
- Include multiple "levels" that the game will progress through
- Include background music

Project Scope

The scope of our project is to create a role-play game with a GUI displaying a top-down view, grid/tile-based levels, pixel art, background music, enemies, a final boss, and various

game items. We will NOT be creating a game that takes multiple hours of progression to fully complete, multiplayer interaction, or internet connectivity.

Team Expectations

As a team, we expect to develop a fully functional two-dimensional role-play game with a main end goal, multiple "levels" to progress through, and minimal bugs/errors.

Coding Standards and Practices Statement

The following standards and practices will be effect throughout the entire project's development:

- The Python programming language will be the only programming language used.
- All directories are named in title-case. `ex: Example_Dir_Name`
- All variables and functions are written in lowercase snake case. `ex) example_var_name, example_function_name(param_1, param_2)`
- Constants are named in uppercase snake case. `ex) EXAMPLE_CONSTANT_NAME`
- Comments must be utilized for each major function.
    - single # for notes and descriptions.
    - triple ### for named sections `ex) ### Section Name ###`
- All functions should have only one clearly-defined purpose, be no longer than ten lines long.
    - Note: Constructors and to-string functions are exempt from this clause.
    - Any other exceptions to this rule must have a clear rationale written in a comment.
- The program should follow proper OOP etiquette
    - private data attributes are only directly referenced within getters and setters
    - Superclass-subclass paradigm is followed if applicable
- Tiles on the game map will be 16x16 pixels
- Sprites (players, NPCs, items) on the game map will be 32x32 pixels

rough idea

graphics

- 2D
- top-down view
- grid/tile-based level design
- pixel-art (externally made assets)

sound design

- background/ambient music

- SFX (e.g. hit sound, coin-grab sound, transition sound)

level design

- Hand-made "milestone" levels
- procedurally-generated "filler" levels between milestone levels

Requirements
Security Requirements

Input Validation:

- All input needs to be validated whether it's out of bounds handling, type checking, etc..
- All data with no input or output (yet) has to be null.
- Try and excepts should be used on most if not all validation checks.
- If a try and except fails, display error message to explain why it failed.
- Maybe try to implement a black box function that checks for inputs.

Unit Tests

- Make sure to try and use unit test from the python libraries to run test.
- Try to create a test with each function (if possible).
- Try to do other ways of unit tests.

GUI

- Make sure inputs are working correctly.
- Make sure user can't crash the GUI by spamming random characters.
- Make sure user can't break the game.

OOP Practices

- Make sure variables are private especially since we are dealing with OOP.
- Make sure we use getters and setters.
- Make sure we use default constructors and constructors.

User Requirements

In order for the developed video game to work, the user will require the following:

- Motor skills such as hand and eye coordination
- The ability to solve problems related to the game's plot
- A computer (desktop/laptop), mouse, and keyboard

Hardware Requirements

In order for the developed video game to work, the following hardware is required:

- A working computer, mouse, and keyboard
- A constant power supply to maintain the computer's operability
- An installation of the most recent Python interpreter

Security

Security Design Document

Assets to Consider

- GUI Main
- Main executable program
- Objects (Getters / Setters / Constructors)

Threats to Assets

Main / GUI Main Executable

- Malicious users
- Improper overwriting
- Pages being scraped
- Improper user input

Objects

- Incorrect use getters / setters / helpers
- Incorrect use of variables (not being private)
- Improper user input
- Malicious Users

Security Measures

Main / GUI Main Executable

- Try and except all input to handle, format, and validate all data correctly
- Input validation black box functions
- User given very limited access to limit any accidental or malicious problems cause by the user
- Clean, check, and return the correct user input matching the variable needed etc...

Objects

- Try and except all input to handle, format, and validate all data correctly (if can be done)
- Input validation make sure setters and getters are correct
- Make sure the variables ARE PRIVATE

Security Summary

Overall, the security we're dealing with mainly comes from the input validation and the clumsiness of the user itself and poorly written code. We must make sure that we validate all functions and try to make sure we use try and catch statements. Our code must be safe with private variables and be clean. Another thing that would be absolutely great just to add is the code being readable so that any person can gloss through it and see which function does what.

Security Statement

As a security team lead, we want to make sure our data is as safe as possible. Data being safe is my main priority as the security lead so we'd like to put measures into place to counter these unsafe practices.

- Make sure we use private variables so outsiders can't access code
- Make sure we validate input so we get the correct input into whatever we need
- Make sure GUI syncs with the main code so there isn't any delay or lag between the two
- Make sure to have error checking ex: __private_variable = -1;
- Try and use try and catch exceptions to remove those pesky exceptions

Overall, let's run unit tests and let's create a safe RPG program where the user can't completely break the game! If we manage to follow the following practices then we'll be able to better the user's enjoyment of our lovely game! We hope to have a great outcome of a fantastic RPG and to do so, we must be able to put security ahead first.

Maintenance

Maintenance Overview

The purpose of performing maintenance on this project is to fix various bugs and performance issues users experience while playing the video game. Additionally, new features and game elements must be added to create interest from seasoned players of the game.

The following types of maintenance will be conducted throughout the software's life-cycle:

- Corrective: fixing any bugs, errors, and glitches that are experienced during gameplay
- Adaptive: allowing the game to remain functional on new, updated, and improved hardware/software environments
- Perfective: constantly improve the performance of the game to maintain an acceptable level of operability
- Preventive: implementing fixes to potential issues, resolving security vulnerabilities, and supporting new computing environments before the become popular enough to cause noticeable problems during gameplay

We will conduct regular testing on our programs to ensure they function as intended. These will be completed before the presentation of a new prototype and will be altered if unsuccessful until an adequate level of success has been reached.

Along with any updates/patches to the game, documentation of these changes, processes, and testing will be completed to communicate progress in the game's life-cycle.

Maintenance Timeline

Week 1 of Development

- Identify initial software, hardware, security, and user requirements
- Conduct testing on any developed elements to ensure proper functionality
- Document changes, testing, and all communication to track progress

Week 2 of Development

- Reevaluate the project's requirements, scope, components, storyline, and other implementation elements to suit the needs of the solution to the identified problem
- Conduct testing on any developed elements to ensure proper functionality

Week 3 of Development

- Finalize initial software, hardware, security, and user requirements
- Conduct final testing on any developed elements to ensure proper functionality

Post-Release

- Note bugs, errors, and glitches (if any) that users encounter during gameplay
- Release updates, security patches, and changes for any identified issues to realign the game with its intended functionality
- Design new elements to add to the game to increase the longevity of seasoned players' interest

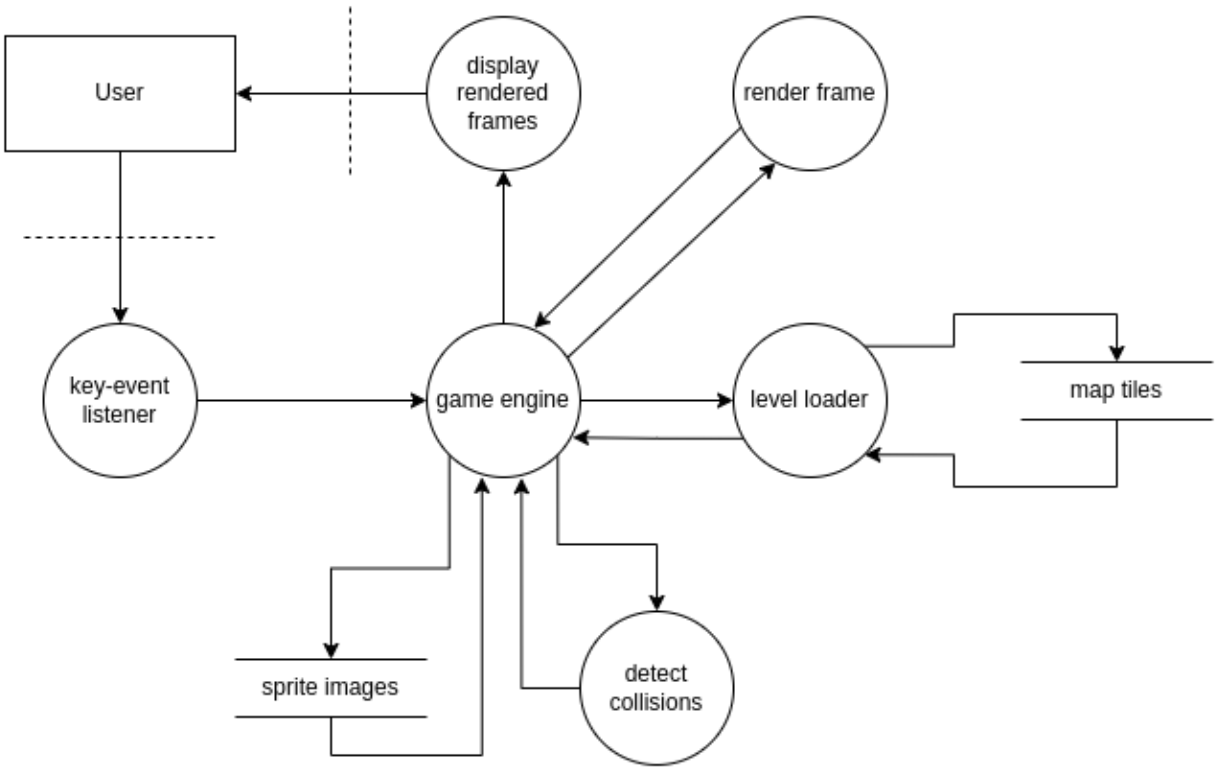
External Asset Credits

Thank you to the following where we sourced our assets (graphics and sound) from:

https://elvgames.itch.io/

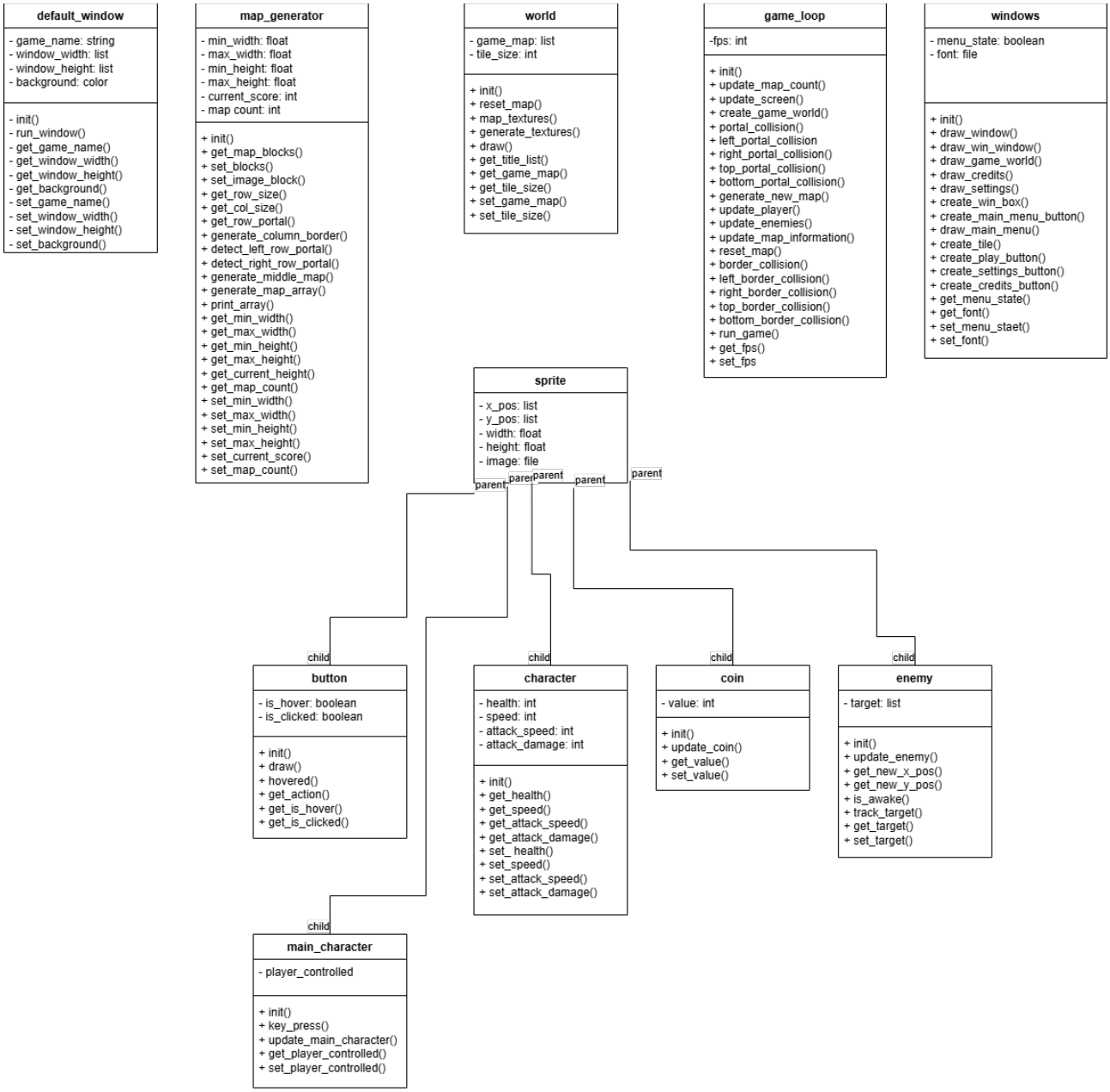
Diagrams

Data Flow Diagram - L1



Problem Frame

UML


default_window
- game_name: string
- window_width: list
- window_height: list
- background: color
- init()
- run_window()
- get_game_name()
- get_window_width()
- get_window_height()
- get_background()
- set_game_name()
- set_window_width()
- set_window_height()
- set_background()
map_generator
- min_width: float
- max_width: float
- min_height: float
- max_height: float
- current_score: int
- map count: int
+ init()
+ get_map_blocks()
+ set_blocks()
+ set_image_block()
+ get_row_size()
+ get_col_size()
+ get_row_portal()
+ generate_column_border()
+ detect_left_row_portal()
+ detect_right_row_portal()
+ generate_middle_map()
+ generate_map_array()
+ print_array()
+ get_min_width()
+ get_max_width()
+ get_min_height()
+ get_max_height()
+ get_current_height()
+ get_map_count()
+ set_min_width()
+ set_max_width()
+ set_min_height()
+ set_max_height()
+ set_current_score()
+ set_map_count()
world
- game_map: list
- tile_size: int
+ init()
+ reset_map()
+ map_textures()
+ generate_textures()
+ draw()
+ get_title_list()
+ get_game_map()
+ get_tile_size()
+ set_game_map()
+ set_tile_size()
game_loop
-fps: int
+ init()
+ update_map_count()
+ update_screen()
+ create_game_world()
+ portal_collision()
+ left_portal_collision
+ right_portal_collision()
+ top_portal_collision()
+ bottom_portal_collision()
+ generate_new_map()
+ update_player()
+ update_enemies()
+ update_map_information()
+ reset_map()
+ border_collision()
+ left_border_collision()
+ right_border_collision()
+ top_border_collision()
+ bottom_border_collision()
+ run_game()
+ get_fps()
+ set_fps
windows
- menu_state: boolean
- font: file
+ init()
+ draw_window()
+ draw_win_window()
+ draw_game_world()
+ draw_credits()
+ draw_settings()
+ create_win_box()
+ create_main_menu_button()
+ draw_main_menu()
+ create_tile()
+ create_play_button()
+ create_settings_button()
+ create_credits_button()
+ get_menu_state()
+ get_font()
+ set_menu_staet()
+ set_font()
sprite
- x_pos: list
- y_pos: list
- width: float
- height: float
- image: file
parent
parent
parent
parent
parent
child
child
child
child
button
- is_hover: boolean
- is_clicked: boolean
+ init()
+ draw()
+ hovered()
+ get_action()
+ get_is_hover()
+ get_is_clicked()
character
- health: int
- speed: int
- attack_speed: int
- attack_damage: int
+ init()
+ get_health()
+ get_speed()
+ get_attack_speed()
+ get_attack_damage()
+ set_ health()
+ set_speed()
+ set_attack_speed()
+ set_attack_damage()
coin
- value: int
+ init()
+ update_coin()
+ get_value()
+ set_value()
enemy
- target: list
+ init()
+ update_enemy()
+ get_new_x_pos()
+ get_new_y_pos()
+ is_awake()
+ track_target()
+ get_target()
+ set_target()
child
main_character
- player_controlled
+ init()
+ key_press()
+ update_main_character()
+ get_player_controlled()
+ set_player_controlled()