Project B.A.T. System Design Document

Last Revised: 25 February 2022

Kevin Green (k_green5@u.pacific.edu)

Project Wiki Page:

https://github.com/comp195/senior-project-spring-2022-blueprint-automation-tool-1/wiki

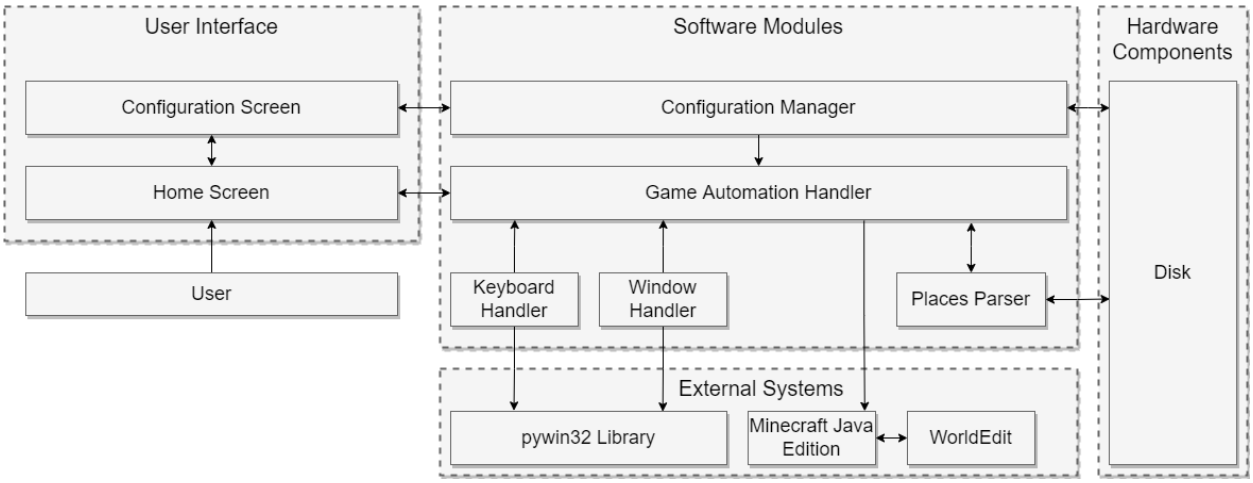
Table of Contents

Introduction

Project B.A.T. is a blueprint automation tool that takes the saved “places” from Google Earth Pro and builds those places in Minecraft to scale. The application takes .kml and .kmz files saved by Google Earth Pro and uses them to create a 2D layout that players can later use to complete the actual building with their creativity.

Project B.A.T. makes uses of reference coordinates and map projection algorithms to handle distortions to the map due to converting places from a ball earth to a perfectly flat Minecraft world. The project also supports multiple blocks to allow users to color-code their layouts.

System Architecture



Software Modules

Configuration Manager

The configuration manager is the module that takes care of updating and saving settings for the program. While the configuration manager handles changing settings in the program, the game automation handler will call to it to get certain settings that it needs that will affect calculations for map projections, scaling, keyboard input timing, and what commands are available for it to use. The configuration manager will also keep track of the last loaded reference point and saved places so that a user can pick up where they left off if they need to take a break in-between use of the program.

Game Automation Handler

The game automation handler module is what ties most of the other software modules together to create a functioning application. Being the main interface with the home screen of the application, this module oversees telling the places parser module what file to read. It then takes the information given to it by the parser and the configuration manager and completes the calculations to convert the geographical coordinates to the coordinate system that Minecraft uses. Once this is figured out, it creates the necessary commands to recreate the place. The automation handler then takes advantage of the keyboard and window handlers to switch to the Minecraft window and execute the commands that it came up with. Afterward, the handler switches back to the blueprint tool's user interface for further user interaction.

Keyboard Handler

The keyboard handler oversees keyboard inputs to Windows to automate the interaction with Minecraft once it is set as the primary screen. Multiple keyboard inputs are needed to complete different interactions such as opening the game's text chat, entering and verifying the commands for World Edit, and teleporting the player around the map. To minimize mistakes, the keyboard handler can implement self-checking to make sure that anything that any text that is inputted into a chat box matches what should have been typed. This should ensure that there are not any mistakes in commands that are given to the game, to avoid unexpected results.

Window Handler

The window handler oversees switching back and forth between the open Minecraft client and the program window during the game automation. This handler needs to be able to successfully find and switch to the Minecraft window so that the program can input the right commands to recreate the places in Minecraft. After automation is complete, the module should

be able to switch back to the blueprint automation tool so that the user can move onto a new saved place or tweak settings if necessary.

Places Parser

The “Places” parser is the module that oversees reading in the places from Google Earth Pro and finding the relevant information that is necessary to convert it into Minecraft blocks. This involves reading in the .kml and .kmz files that are the actual files that are created when a place is saved in Google Earth Pro. Then the parser reads through the file to find any relevant information related to the name of the place, the shape of the place, and the geographical coordinates that are saved within the files.

User Interfaces

Configuration Screen

The configuration screen of the user interface is where the user will primarily make changes to how the program will perform when executing the automation steps. From this screen, the user makes changes to the scale of the layouts built, the preferred conversion method from geographical coordinates to Minecraft coordinates, and the default height level that blueprints should be built on.

Home Screen

The home screen in the user interface is the main interface that users of the blueprint automation tool will use. On this screen, users will choose the .kml and .kmz file that they want to have built, the reference point that the build should be centered around, and the block that they want the build to be made from. There will also be a button that will allow the user to start the automation, and to cancel it mid-way if something unwanted occurs as a failsafe.

Hardware, Software, and System Requirements

Hardware Requirements

Systems that plan to run the Blueprint Automation Tool should meet the minimum requirements to run Minecraft as listed by Mojang on their website. Depending on the size of the blueprint that the user is building, requirements will need to be higher than these listed requirements to experience improved performance in-game while the layouts of the places are being built.

Hardware	***Minimum Requirement***
CPU	Intel Core i3-3210 3.2 GHz AMD A8-7600 APU 3.1 GHz
GPU	*Integrated Graphics:* Intel HD Graphics 4000 (Ivy Bridge) AMD Radeon R5 series (Kaveri line) *Discrete GPU:* NVIDIA GeForce 400 Series AMD Radeon HD 7000 series
RAM	4 GB
Disk Space	1 GB
Internet	Internet is required for the initial installation of the required software. Afterward, offline use of the tool is possible if the user is not connecting to an online multiplayer server.

Software Requirements

Software	***Minimum Version***
Google Earth Pro	Latest available version
Minecraft Java Edition	Version 1.12.0 or newer
WorldEdit	Version 6.1.8 (for Minecraft 1.12) or newer
Java	Version 8 or newer

Note: Java 8 will automatically be installed for Minecraft versions 1.12 and newer when the game is installed onto the system. Multiplayer servers will need Java 8 to be manually installed on the host server to run.

System Requirements

System	***Supported Versions***
Windows	Windows 10 and 11

External Interfaces

pywin32 Python Library

The pywin32 library for Python is a series of modules that allow access to Windows APIs in python. In this project, this library will be used extensively to allow keyboard and window automation to automate the process of building the scaled blueprints within the Minecraft environment.

More information and documentation for pywin32 are available through their online help file found at https://mhammond.github.io/pywin32/.

Minecraft Java Edition

Minecraft is a popular video game where a player has a nearly infinite world to explore and build in. Java Edition is the original version of the game that is specific to desktop computers. This project is focusing on the creative version of the game, where players have infinite resources to build anything they like. Project B.A.T. takes advantage of these gameplay features to quickly create to-scale layouts of real-life locations.

More information and documentation about Minecraft, including gameplay mechanics, tutorials, changelogs, and available commands can be found in the community ran wiki for the game at https://minecraft.fandom.com/wiki/Minecraft_Wiki.

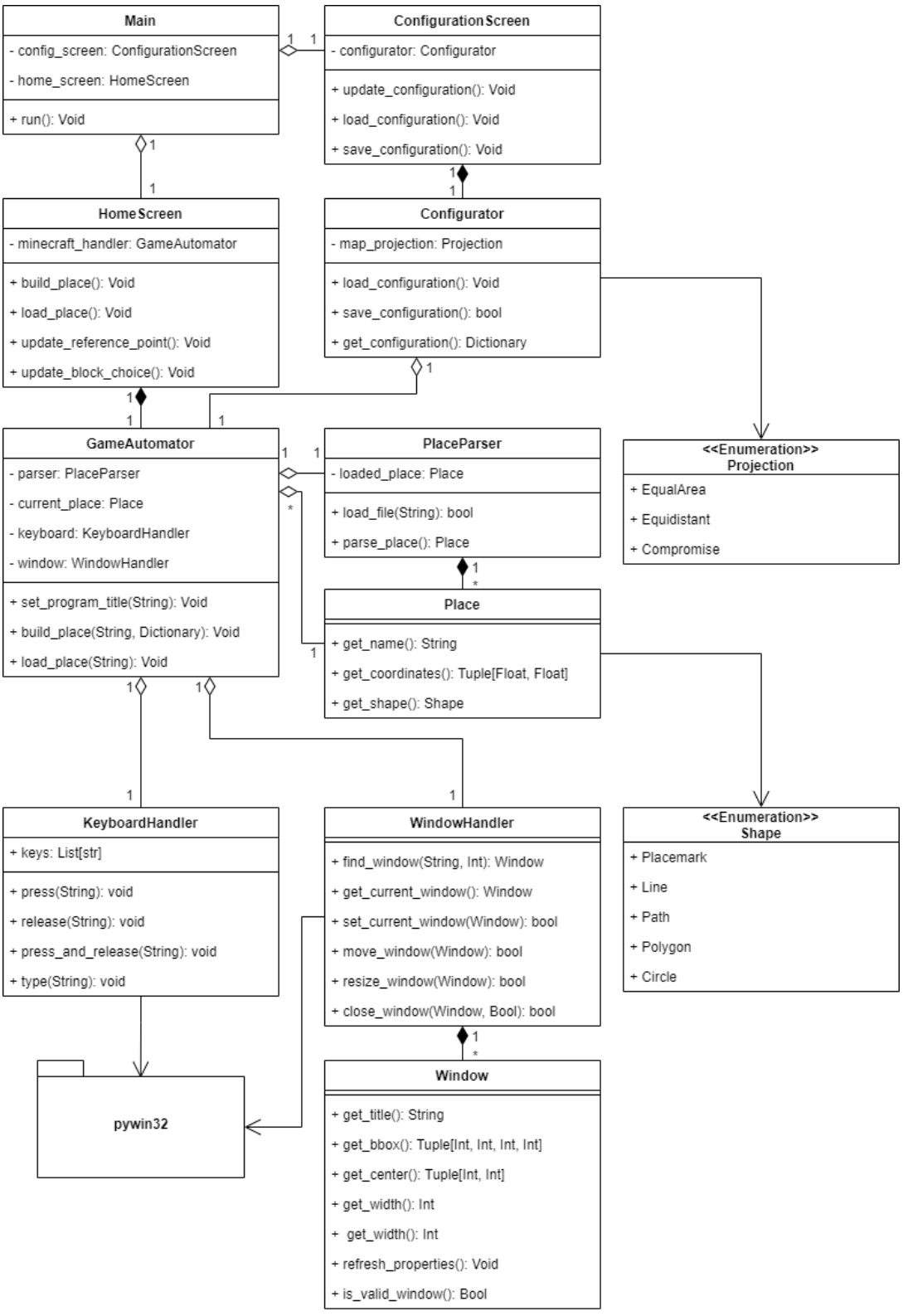
WorldEdit

WorldEdit is a plugin and mod to the base vanilla version of Minecraft that allows players to quickly build large structures in-game without having to place each block individually using commands that can be entered through the text chat. WorldEdit is being used in this project to quickly create the layouts of places without having to manually set each block using vanilla commands.

More information and documentation for WorldEdit and its available commands can be found at its website at https://worldedit.enginehub.org/en/latest/.

Software Design

Class Diagram



Class Specifications

User Interface

Main
- config_screen: ConfigurationScreen - home_screen: HomeScreen
- switch_to_home(): void - switch_to_config(): Void - start_program(): void - close_program(): void + run(): Void

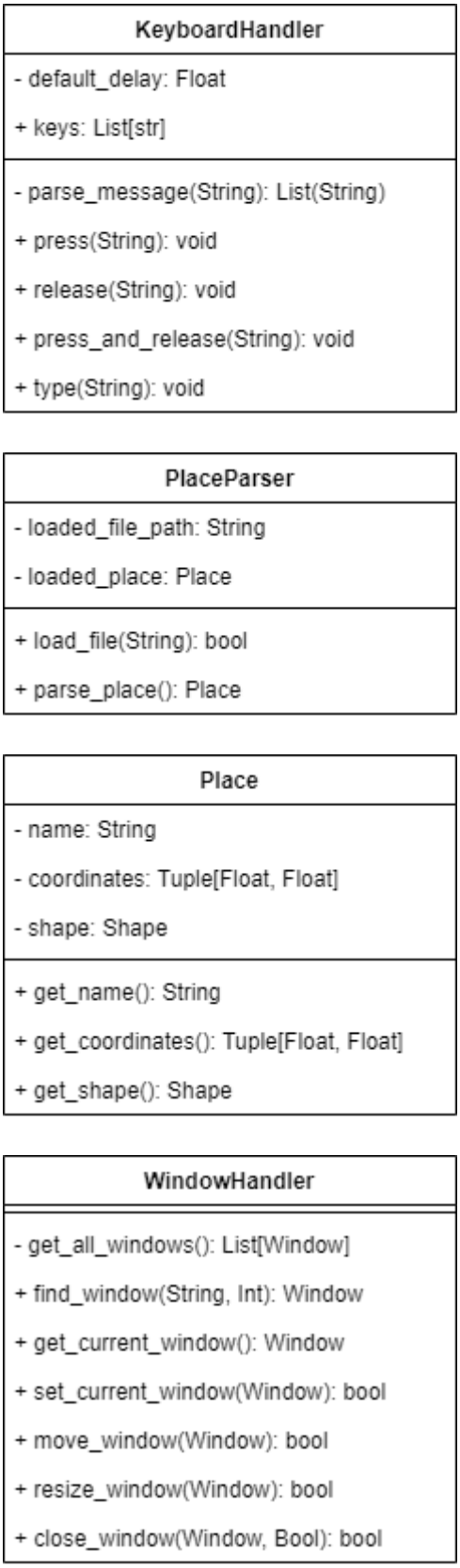
ConfigurationScreen
- configurator: Configurator
+ update_configuration(): Void + load_configuration(): Void + save_configuration(): Void

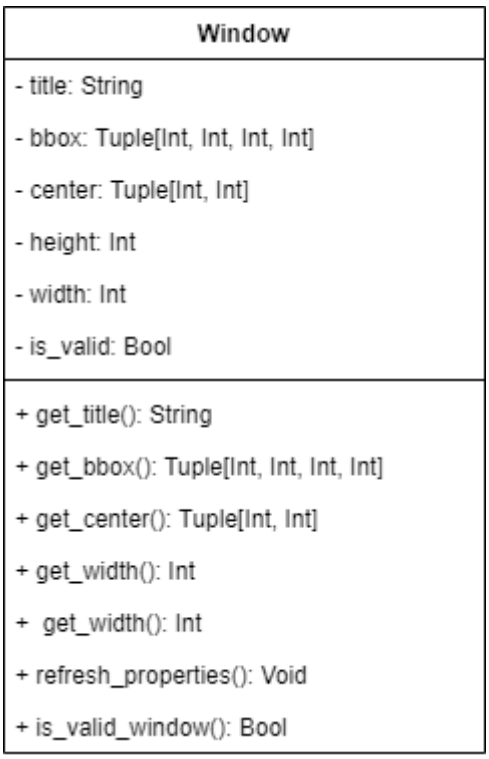
HomeScreen
- minecraft_handler: GameAutomator - current_block_choice: String
+ build_place(): Void + load_place(): Void + update_reference_point(): Void + update_block_choice(): Void

Software Modules

Configurator
- settings_file_path: String - map_projection: Projection - scale_factor: Float - starting_height: Int - last_saved_place: Path - last_saved_reference: Path
+ load_configuration(): Void + save_configuration(): bool + get_configuration(): Dictionary

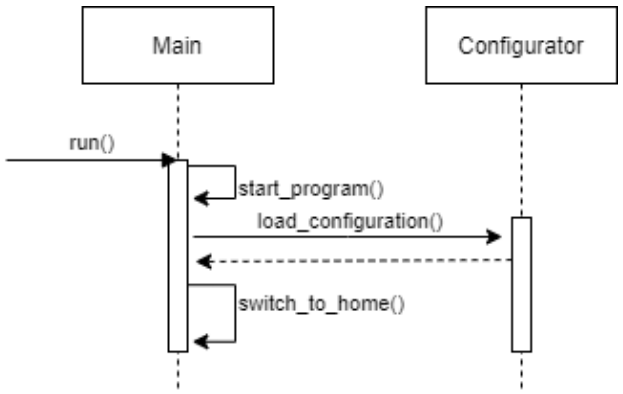
GameAutomator
- game_window_title: String - current_place: Place - program_window_title: String - keyboard: KeyboardHandler - window: WindowHandler - parser: PlaceParser
- get_place_information(): Dictionary - convert_coordinates(): Tuple[Int, Int, Int] - convert_equidistant(): Tuple[Int, Int, Int] - convert_equal_area(): Tuple[Int, Int, Int] - convert_compromise(): Tuple[Int, Int, Int] - find_game_version(): String - execute_command(String): Void + set_program_title(String): Void + build_place(String, Dictionary): Void + load_place(String): Void


KeyboardHandler
- default_delay: Float
+ keys: List[str]
- parse_message(String): List(String)
+ press(String): void
+ release(String): void
+ press_and_release(String): void
+ type(String): void
PlaceParser
- loaded_file_path: String
- loaded_place: Place
+ load_file(String): bool
+ parse_place(): Place
Place
- name: String
- coordinates: Tuple[Float, Float]
- shape: Shape
+ get_name(): String
+ get_coordinates(): Tuple[Float, Float]
+ get_shape(): Shape
WindowHandler
- get_all_windows(): List[Window]
+ find_window(String, Int): Window
+ get_current_window(): Window
+ set_current_window(Window): bool
+ move_window(Window): bool
+ resize_window(Window): bool
+ close_window(Window, Bool): bool

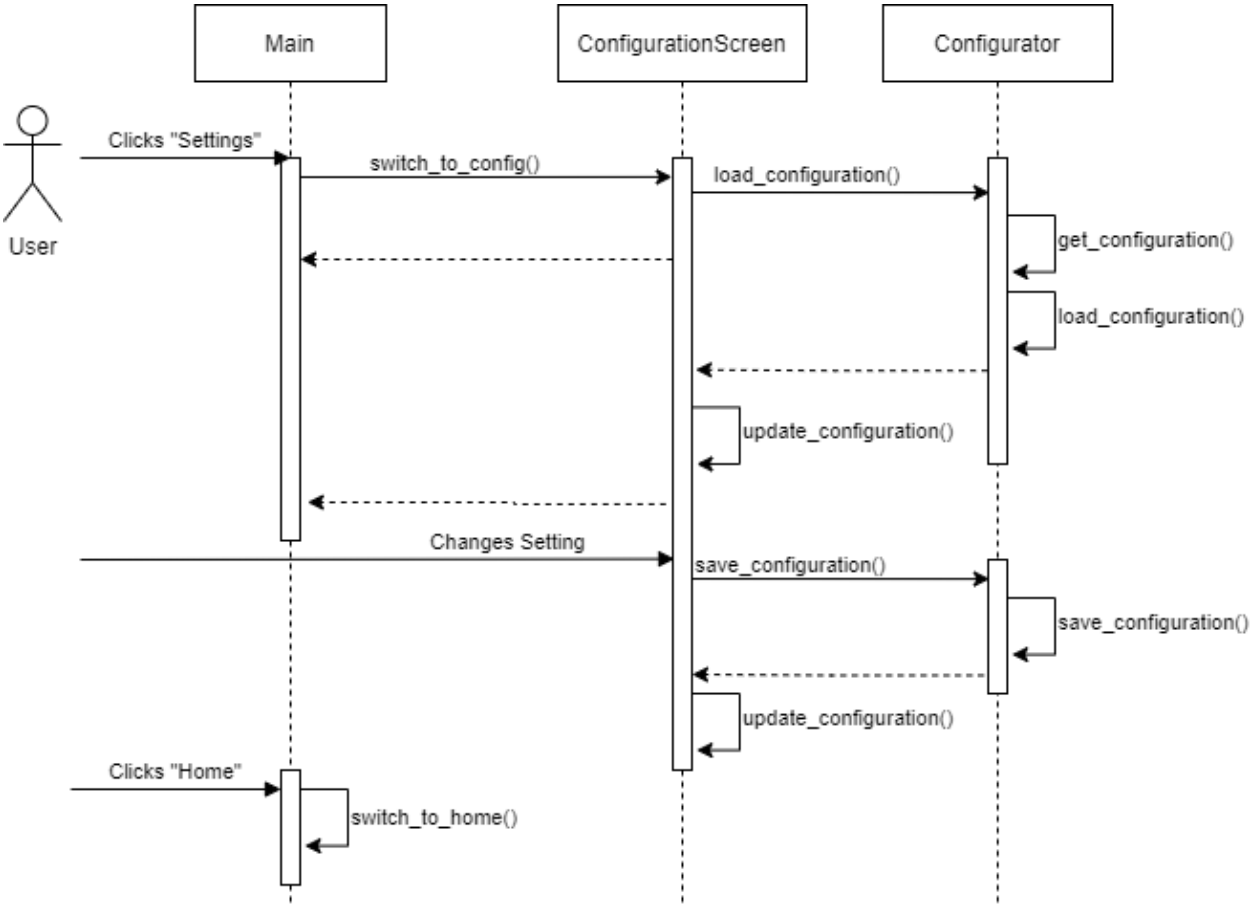

Window
- title: String
- bbox: Tuple[Int, Int, Int, Int]
- center: Tuple[Int, Int]
- height: Int
- width: Int
- is_valid: Bool
+ get_title(): String
+ get_bbox(): Tuple[Int, Int, Int, Int]
+ get_center(): Tuple[Int, Int]
+ get_width(): Int
+ get_width(): Int
+ refresh_properties(): Void
+ is_valid_window(): Bool

Interaction Diagrams

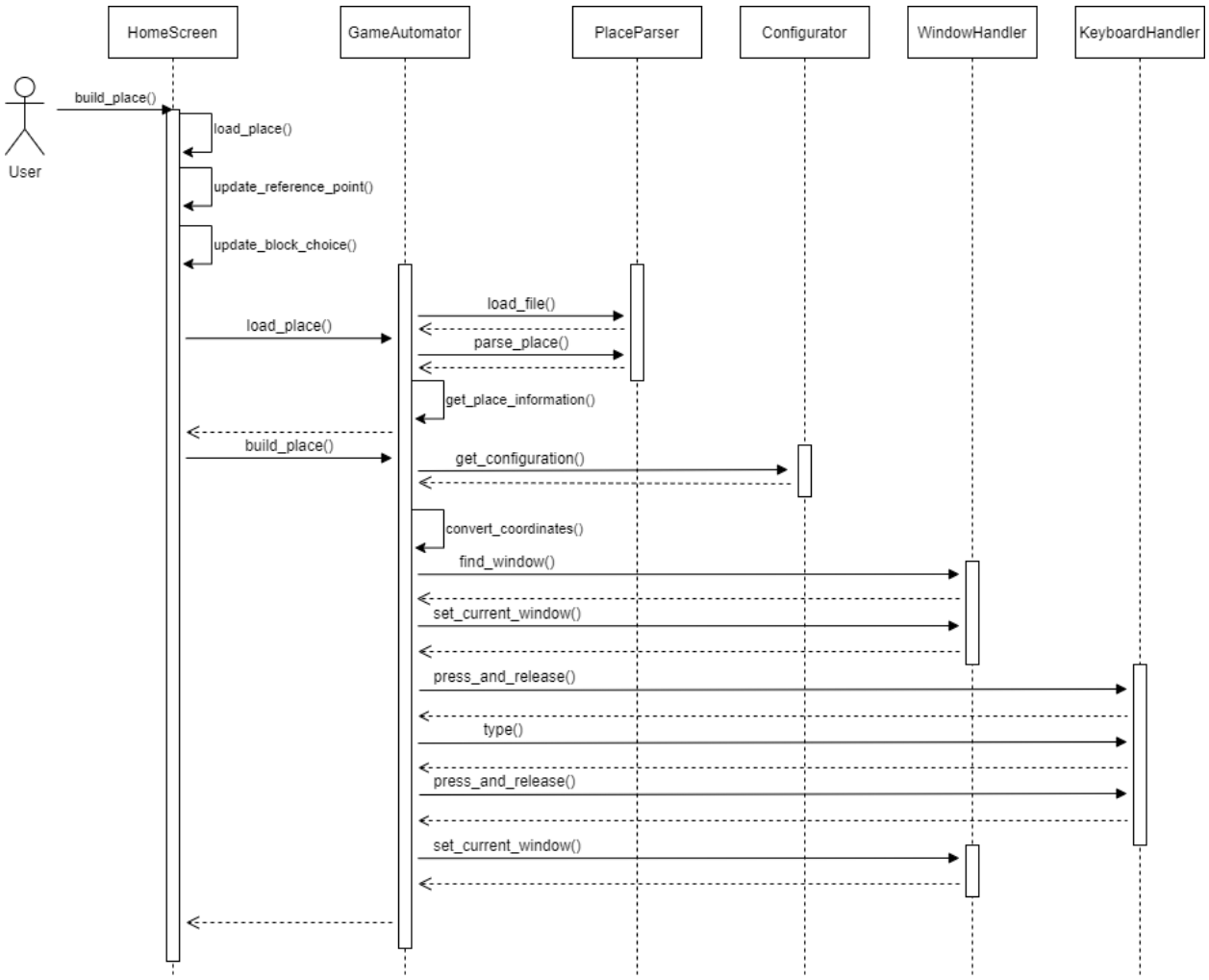
Starting the Program



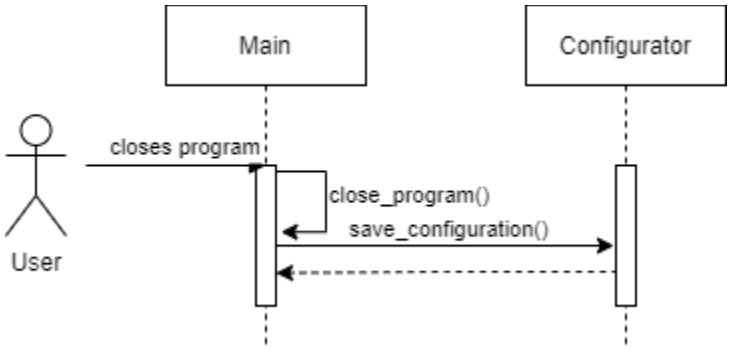
Updating Configurations



Building A Saved Place



Stopping the Program



Design Considerations

Modularity

Classes for this project have been designed in a way that each class handles one specific area of the automation process. For example, the only job of the WindowHandler class is to handle window navigation on the screen, which involves bringing windows into focus. Another example is that the KeyboardHandler class is solely responsible for keyboard inputs to the operating system. Designing the project in this manner allows for easier debugging when something in the project behaves inconsistently or incorrectly, since the location of the code where the bug could be should theoretically only exist in one place. Additionally, the design of these "modules" allows them to be easily swapped out later for better versions of the same functionality. This design also will allow for the modules to be reused in future projects relatively easy.

User Simplicity

One of the main goals for Project B.A.T. is to be as simple for the user to use as possible, given that they have all the required software installed. Using only two screens with minimal user input required, the application should become more easily understandable while having enough features and capabilities to allow the tool to be faster than manually creating the layouts by hand.

Automation Speed and Accuracy

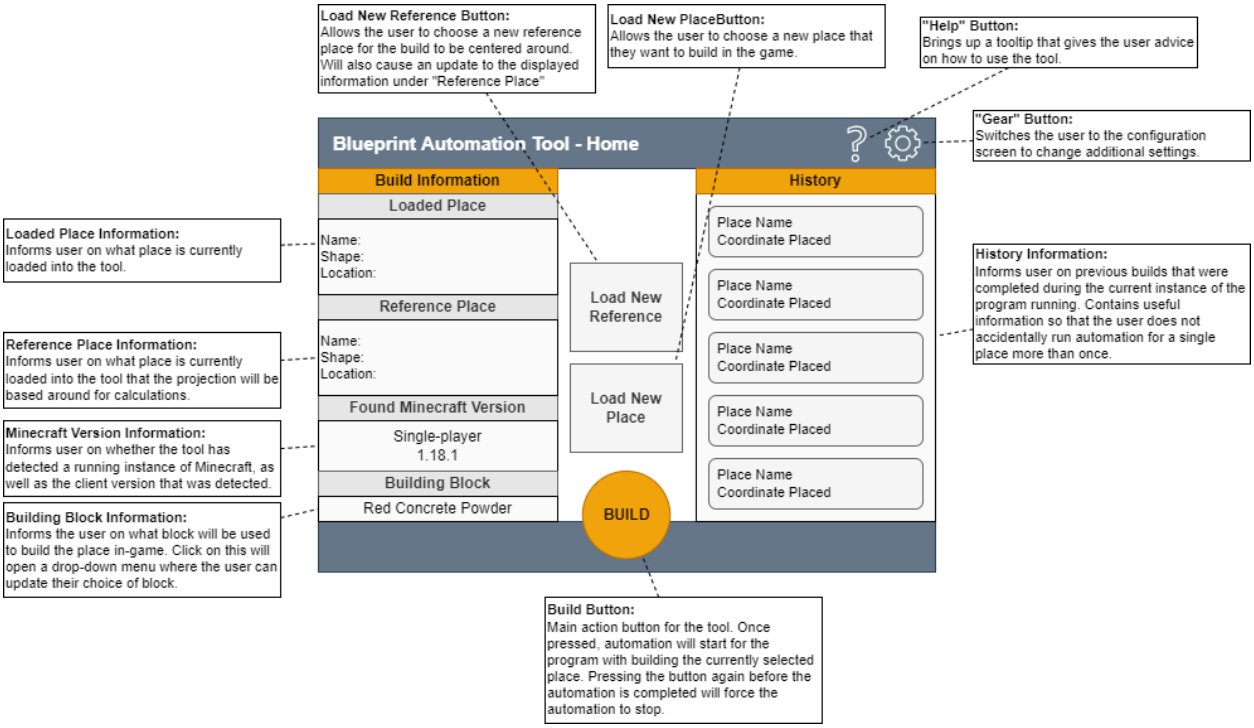
If Project B.A.T. takes too long to successfully recreate layouts in saved places within Minecraft, it has no major advantage over just manually building the layouts by hand. Therefore, automation classes were designed to take advantage of more low-level libraries for sending calls

to the API to avoid the overhead that can be caused by other automation libraries that already exist for Python.

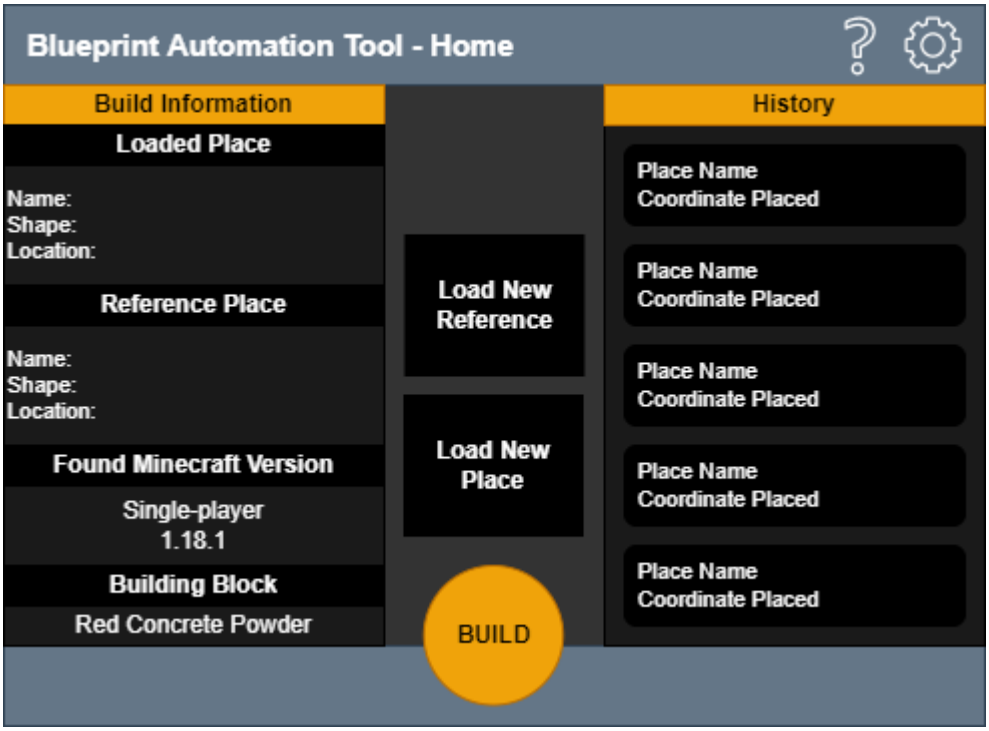
However, if the automation is not also accurate, then disastrous results could ensue. This accuracy problem is also a major consideration in why the creation of custom automation handlers of keyboards and window management is necessary. These custom modules allow for more controllable error detection and fail-safe checking that will allow for safer use of the project without major risks that could arise due to any number of system and user variables that cannot always be accounted for.

User Interface Designs

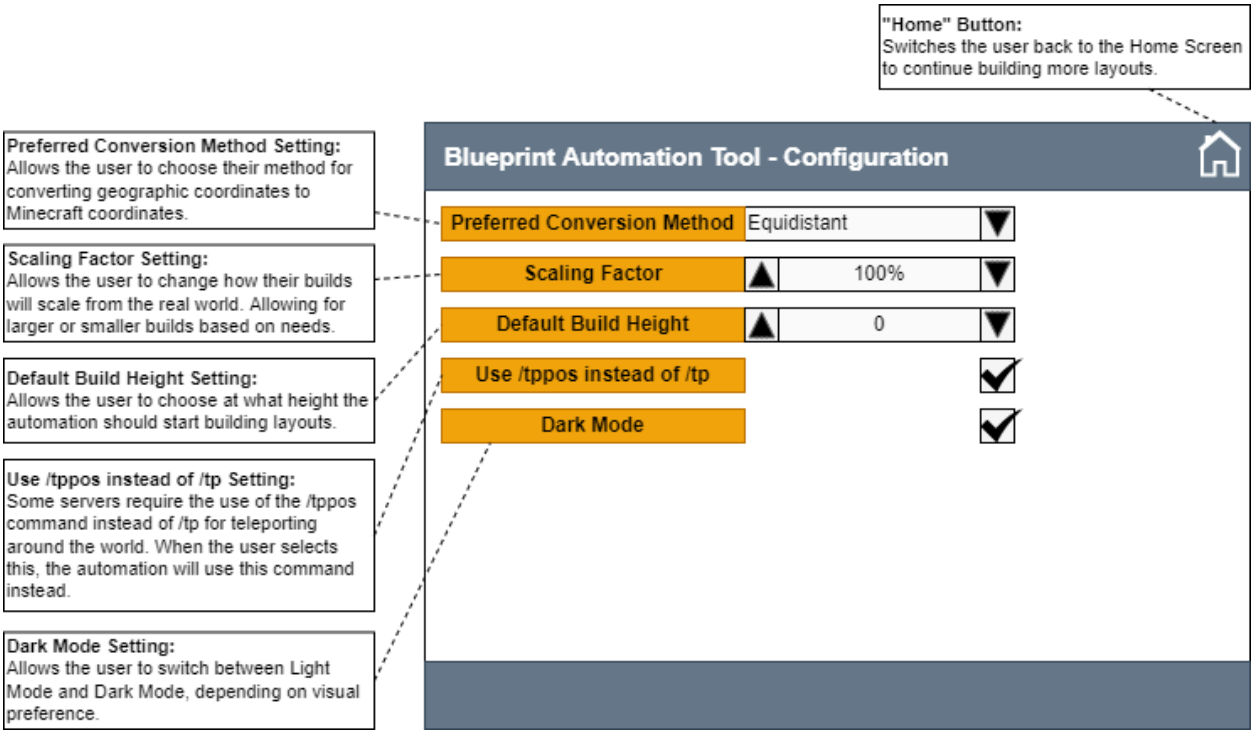
Home Screen



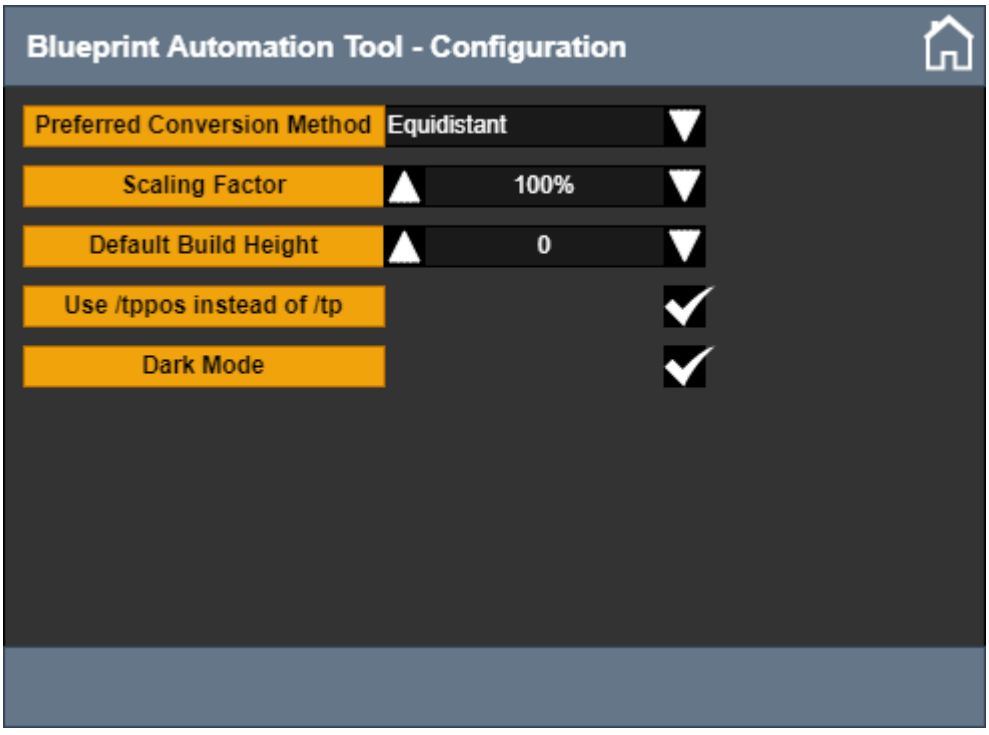
Dark Mode Variant



Configuration Screen



Dark Mode Variant



Glossary of Terms

Plugin:

An additional extension added to a Minecraft that is installed onto a dedicated server. These extensions do not alter any gameplay mechanics themselves but only add onto existing ones.

Mod:

(Short for modification) Third-party changes or extensions of the original gameplay mechanics which can be installed onto single-player or multiplayer versions of the game.

Vanilla:

Refers to a video game that is in its original state. No mods or plugins have been installed to change or extend gameplay mechanics.

Block:

The basic unit of structure in Minecraft. Blocks make up the entire world, each block being equivalent to exactly one meter in the real world.

Text Chat:

The main form of communication between Minecraft players in the same world. This text box allows players to enter messages that will be shown to the other players in the world. If cheats are enabled on a server or single-player world, players can enter commands that will result in various effects taking place towards other players or the world.

Place:

A saved location from Google Earth Pro that is saved as a .kml or .kmz file that is located locally on a user's computer.

References

BuildTheEarth, “Build The Earth,” *BuildTheEarth,* 2022. [Online]. Available: https://buildtheearth.net/. [Accessed: 19-Jan-2022].

EngineHub, “WorldEdit 7.2 Documentation,” *WorldEdit Documentation,* 2022. [Online]. Available: https://worldedit.enginehub.org/en/latest/. [Accessed: 02-Feb-2022].

EngineHub, “WorldEdit,” *EngineHub*. [Online]. Available: https://enginehub.org/worldedit/. [Accessed: 19-Jan-2022].

Google, “Earth versions – google earth,” *Google*, 2022. [Online]. Available: https://www.google.com/earth/versions/. [Accessed: 19-Jan-2022].

H. Goebel, G. Bajo, D. Vierra, D. Cortesi, and M. Zibricky, “pyinstaller,” *PyPI,* 2022. [Online]. Available: https://pypi.org/project/pyinstaller/. [Accessed: 19-Jan-2022].

ImagineFun, “Imagine fun,” *Imagine Fun*, 2021. [Online]. Available: https://imagineeringfun.net/. [Accessed: 19-Jan-2022].

M. Hammond, “Python for Win32 Extensions Help,” *GitHub.io*, 2022. [Online]. Available: https://mhammond.github.io/pywin32/. [Accessed: 02-Feb-2022].

M. Hammond, “pywin32,” *PyPI,* 2022. [Online]. Available: https://pypi.org/project/pywin32/. [Accessed: 19-Jan-2022].

MCParks, *MCParks,* 2022. [Online]. Available: https://mcparks.us/. [Accessed: 19-Feb-2022].

“Minecraft Wiki,” *Minecraft Wiki – The Ultimate Resource for Minecraft*, 2022. [Online]. Available: https://minecraft.fandom.com/wiki/Minecraft_Wiki. [Accessed: 02-Feb-2022].

Mojang, “Get minecraft,” *Minecraft.net,* 21-Sep-2021. [Online]. Available: https://www.minecraft.net/en-us/get-minecraft. [Accessed: 19-Jan-2022].

Mojang, “Minecraft official site,” *Minecraft.net*, 10-Nov-2021. [Online]. Available: https://www.minecraft.net/en-us. [Accessed: 19-Jan-2022].

Python Software Foundation, “Welcome to Python.org,” *Python.org*, 2022. [Online]. Available: https://www.python.org/. [Accessed: 19-Jan-2022].