What “IF” Machine

A Scriptable Interactive Fiction Engine

Anthony Catricala
Michael Johnson
Dakota Jones
Pat Modica
Lucy Oliverio

User Manual

Table of Contents

What is the What “IF” Machine?

The What IF Machine is a tool that makes it easy to create your own Interactive Fiction games. Our tool is different from other tools on the market in that our tool allows users to write their own Python scripts to create their own functionality to their stories. This could include adding puzzles to their dungeons, giving enchanted items special effects, playing a sound when the player defeats an enemy, opening a webpage to share high scores, or anything you can program!

In addition to being powerful enough for experienced programmers, our tool is easy to use without any programming experience. The What IF Machine ships with many built-in commands, tools, tutorials, and a detailed error handling system to guide new users through making their own interactive worlds.

Getting Started

❖ Downloading the What IF Machine

The What IF Machine can be downloaded from GitHub at https://github.com/AnthonyCatricala/text-engine-4. Simply download it as a .zip file, and extract anywhere to begin using it.

❖ Other Tools Needed

Working in our tool requires the use of a text editor. While most operating systems come with one (Notepad, Gedit), we recommend downloading one better suited to programming, such as Notepad++ or Atom.

Getting Started

❖ Creating a new project

In the What IF Machine, a project is an interactive fiction, a text-based adventure where the reader is placed into the story as the main character and can influence the world around them. Projects are stored as python files with the extension ".py". To create a project, simply create an empty Python file with the name of your new project. For instance, our demo adventure project file is named Demo_Adventure.py. Once you have a project file, you can start building your world by using our API functions or creating environments by hand using the templates.

❖ Loading and playing a game

After your world is built using our API, you can begin playing by running main.py with the the room_name flag and the name of the room your adventure will start in.

Command Example:

python3 main.py --room_name "Starting Room"

Built-In Features

❖ Help Menu

Projects built with the What IF Machine have a built-in help menu that allows the player to quickly view all of the commands they can enter into the game. This menu can be reached at any time while playing a game by typing “help” in the command line.

❖ Error Handling

For users just learning to program, or even just new to our world creation API (Application Programming Interface), we have implemented a detailed error handling system that will find when API function calls are being used inappropriately, explain in plain English exactly where the issue lies, and give the option to open up the documentation for that specific function. This will help new users quickly become familiar with the capabilities of our API, and even allow users to debug their projects before release.

Built-In Features

❖ Player Commands

Our tool comes with a collection of built-in functions to allow a player to easily navigate the world around them. These built-in functions are listed in the chart [below], and are accessible while playing any What IF Machine project by typing “help” in the command line.

❖ Room templates

In addition to using our API functions to create rooms for use in you own games, rooms can also be created and edited manually in a text editor. To make this easier, we have designed an intuitive template for rooms that is intuitively readable and editable to allow non-programmers to quickly and easily create rooms and environments for their projects.

Built-In Features

❖ Triggers

Triggers are special functions that can be written that start special events when certain parts of the world are interacted with. These can be as simple as printing something to the terminal, or unlocking a door, blocking off a pathway, or other features.

❖ Using user-written scripts

Experienced programmers can add extended functionality to the What IF Machine by applying their own custom Python scripts to rooms, locks, items, and other objects in their world. This could mean adding a lock-picking minigame to a certain locked door, having the player complete puzzles to progress through a room, or adding sound effects or opening your website after the credits roll at the end of your story.

API Function Documentation

change_room_name()

Updates a rooms name.

This function takes 2 parameters:

- The room that will be renamed (Room)
- The new name of that room (String)

set_room_description()

Adds a room description to a room.

This function takes 2 parameters:

- The room that will be edited (Room)
- The new description of that room (String)

change_room_description()

Adds a room description to a room.

This function takes 2 parameters:

- The room that will be edited (Room)
- The new description of that room (String)

add_light_to_room()

Adds light to a given room

This function takes a single parameter:

- The room that will be edited. (Room)

remove_light_from_room()

Removes light from a given room

This function takes a single parameter:

- The room that will be edited. (Room)

Create_room()

Creates a new room.

This function takes a single parameter:

- The name of the new room (String)

API Function Documentation

create_door()

Creates a door.

This function takes 4 parameters:

- If the door is open (Boolean)
- If there is a lock on the door (Lock)
- If there is a trigger on the door (Trigger)
- If there are user scripts to activate when using the door (UserScript)

load_room()

Loads an existing room file into the game.

This function takes two parameters:

- The name of the room to be loaded (String)
- The file containing the room to be loaded (.Room file)

save_room()

Saves changes to a room into its .room file

This function takes one parameter:

- The room to be saved (Room)

add_object_to_room()

Adds an object into a given room.

This function takes two parameters:

- The room the object will be added to (Room)
- The object that will be added to the room (Object)

remove_object_from_room()

This function takes two parameters:

- The room that is being edited (Room)
- The object that will be removed (Object)

API Function Documentation

create_lock_and_key()

Creates a lock and key.

This function takes 5 parameters:

- The name of the new key (String)
- The description of the new key (String)
- If the lock is locked by default (Bool)
- Triggers attached to the lock (Trigger)
- User-written scripts attached to the lock (UserScript)

apply_door_to_exit()

Applies a door to an exit, if it was created without one.

This function takes 2 parameters:

- The exit that the door will be applied to (Exit)
- The door that will be applied to the exit (Door)

remove_door_from_exit()

Removes a door from an exit, if it has one.

This function takes 2 parameters:

- The exit that will be edited (Exit)
- The door that will be removed from the exit (Door)

apply_lock_to_door()

Applies a lock to a door.

This function takes 2 parameters:

- The door that the lock will be applied to (Door)
- The lock that will be applied to the door (Lock)

remove_lock_from_door()

Removes a lock from a door that it is applied to.

This function takes 2 parameters:

- The door that the lock will be applied to (Door)
- The lock that will be applied to the door (Lock)

API Function Documentation

create_room_exit()

Creates a possible exit from the room, two of these can be connected with the link_two_rooms() function, or with the second parameter of this function, if the second exit already exists.

This function takes 7 parameters:

- The direction of the exit in the room (String)
- Where the exit connects to (Exit)
- If the exit is blocked by default (Bool)
- If there is a door on the exit (Door)
- If there is a trigger attached to the exit (Trigger)
- If there is a user-written script attached to the exit (UserScript)

apply_exit_to_room()

Applies an exit to a room.

This function takes 2 parameters:

- The room that the exit will be applied to (Room)
- The exit that will be applied to the room (Exit)

remove_exit_from_room(room=None, compass_direction=None):

Removes an exit from the room.

This function takes 2 parameters:

- The room that the exit will be removed from (Room)
- The exit that will be removed from the given room (Exit)

API Function Documentation

create_user_script(trigger_command="", before="", instead="", after=""):

Creates a UserScript object.

Created scripts work by listening for a certain command. When this command is run, a script can execute before or after that command is executed, execute instead of the command, or any of the 3. Python code to be executed can be written as strings in the function call, or as separate .py files linked to in the call.

This function takes 7 parameters:

- The command that will launch the script (String)
- Python code that will be run before the command (String)
- Python code that will be run instead of the command (String)
- Python code that will be run after the command (String)
- Python file that will be run before the command (File)
- Python file that will be run instead of the command (File)
- Python file that will be run after the command (File)

apply_user_script_to_room(room=None, user_script=None):

Applies a user script to a room.

This function takes 2 parameters:

- The room that the script will be added to (Room)
- The UserScript that will be added to the room (UserScript)

apply_user_script_to_door()

Applies a user script to a door

This function takes 2 parameters:

- The door that the script will be added to (Door)
- The UserScript that will be added to the door (UserScript)

API Function Documentation

apply_user_script_to_lock()

Applies a user script to a lock

This function takes 2 parameters:

- The lock that the script will be added to (Lock)
- The UserScript that will be added to the lock (UserScript)

apply_user_script()

Applies a user script to any game object.

This function takes 2 parameters:

- The object that the script will be added to (Object)
- The UserScript that will be added to the object (UserScript)

apply_trigger()

Applies a trigger to any game object.

This function takes 2 parameters:

- The object that the trigger will be added to (Object)
- The trigger object to be applied (Trigger)

link_two_rooms(from_room: Room, to_room: Room, direction: str="", description: str="")