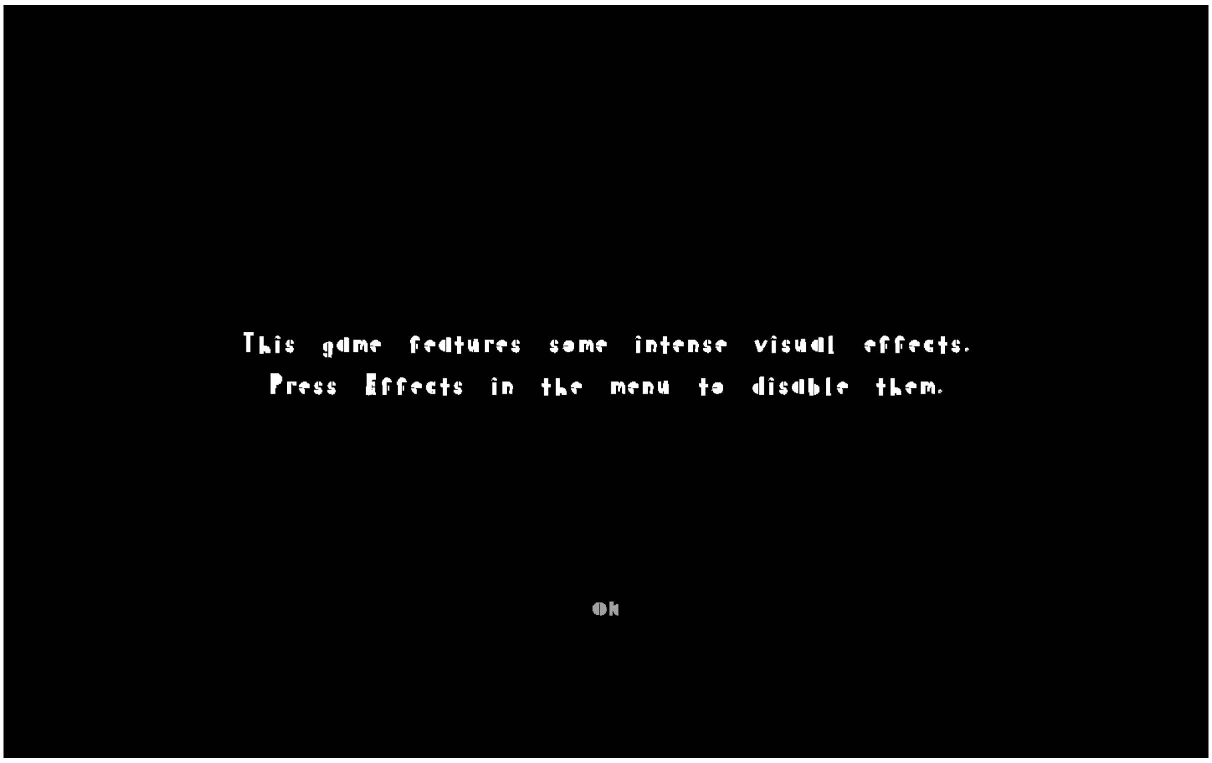
Introduction

Heat of the Moment is an intense, fast-paced game where you must make quick decisions to escape a burning house. As you pass through scenes, you will find items that open up new options and encounter characters like the arsonist and the assassin. Each run will result in one of 21 possible endings. The game is highly replayable and I encourage replaying it many times to see how many different endings you can get. It also features a cross-game storyline with time travel, elementals, and an alternate dimension. This storyline unlocks new items, characters, scenes, and endings; most notably, the True Ending.

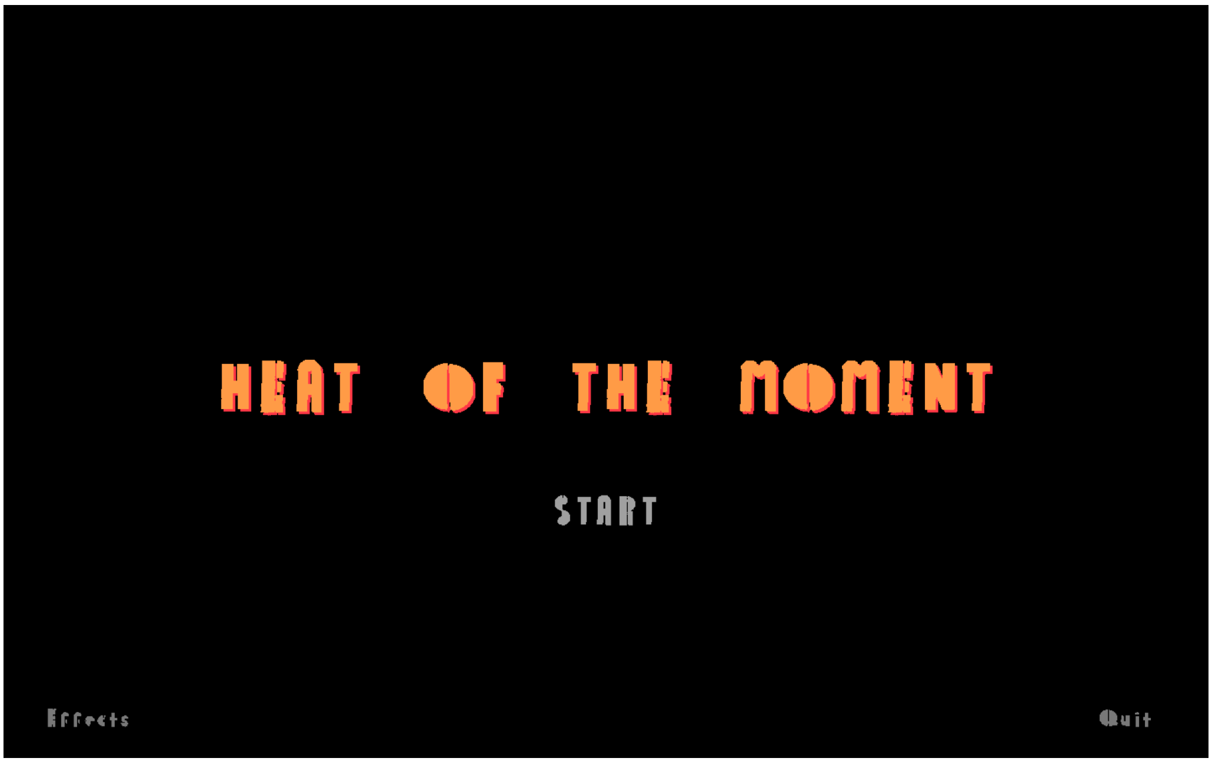
The game's low-resolution visual and audio style is meant to represent the short amount of time and attention the player character has in each scene as well as help the player focus on making choices. Effects like shaking, flashing text and fading to red convey the danger of the situation. *Ape Out* was the primary influence for the project, inspiring elements like the fast-paced gameplay, shaking text, and learning more about where to go and what to do with each game.

Implemented Functionality

Opening the Game


This game features some intense visual effects.
Press Effects in the menu to disable them.

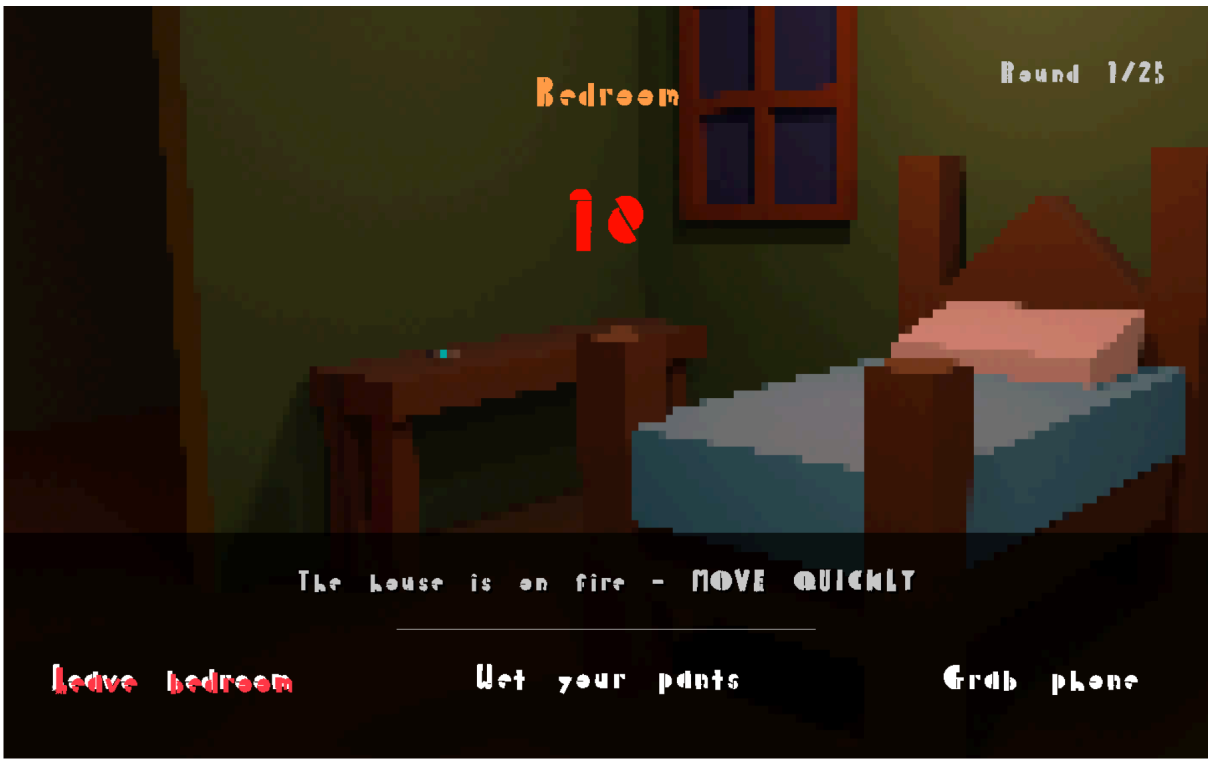
Warning screen



Menu

On opening the game, a screen will display a warning about the intense visual effects in the game. On pressing “Ok”, you will be taken to the menu, which displays the title of the game and some options. From here, you can start the game, toggle intense visual effects with the “Effects” option, and quit the game. I recommend leaving effects on unless they cause discomfort.

Scenes and Options

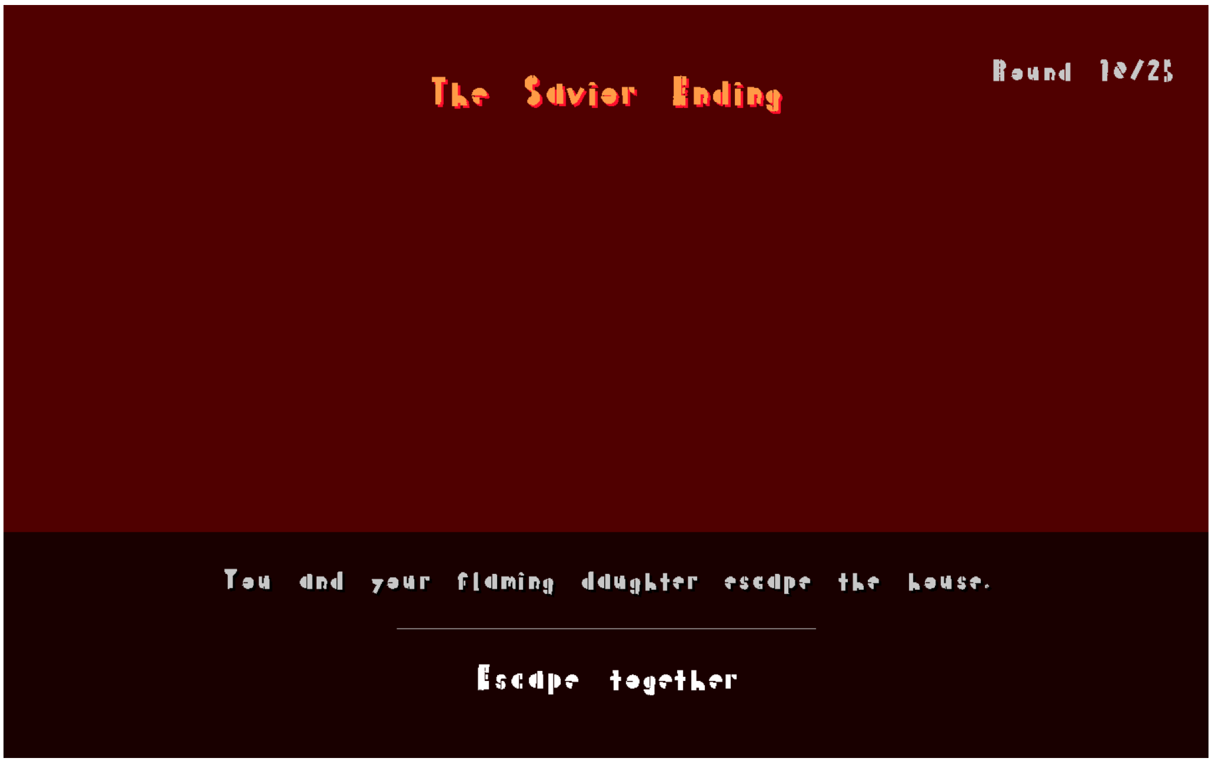


The house is on fire - MOVE QUICKLY

One of the first scenes of the game

Players progress through the game by navigating through scenes. Scenes display an image of the current area the player character is in, a description of the current situation, and a timer displaying how long the player has to choose an option. If the player fails to make a choice by the time the timer runs out, the game will end.

Most actions take a round. Rounds are independent of the timer; no matter how long you idle in a scene, a round will not pass until you take action. Every time a round passes, characters move around the house. If the round counter passes the final round, the house will fall and the game will end. The final round can be pushed back by calling the firefighters and/or obtaining the elemental ring.


The Savior Ending

Round 10/25

You and your flaming daughter escape the house.

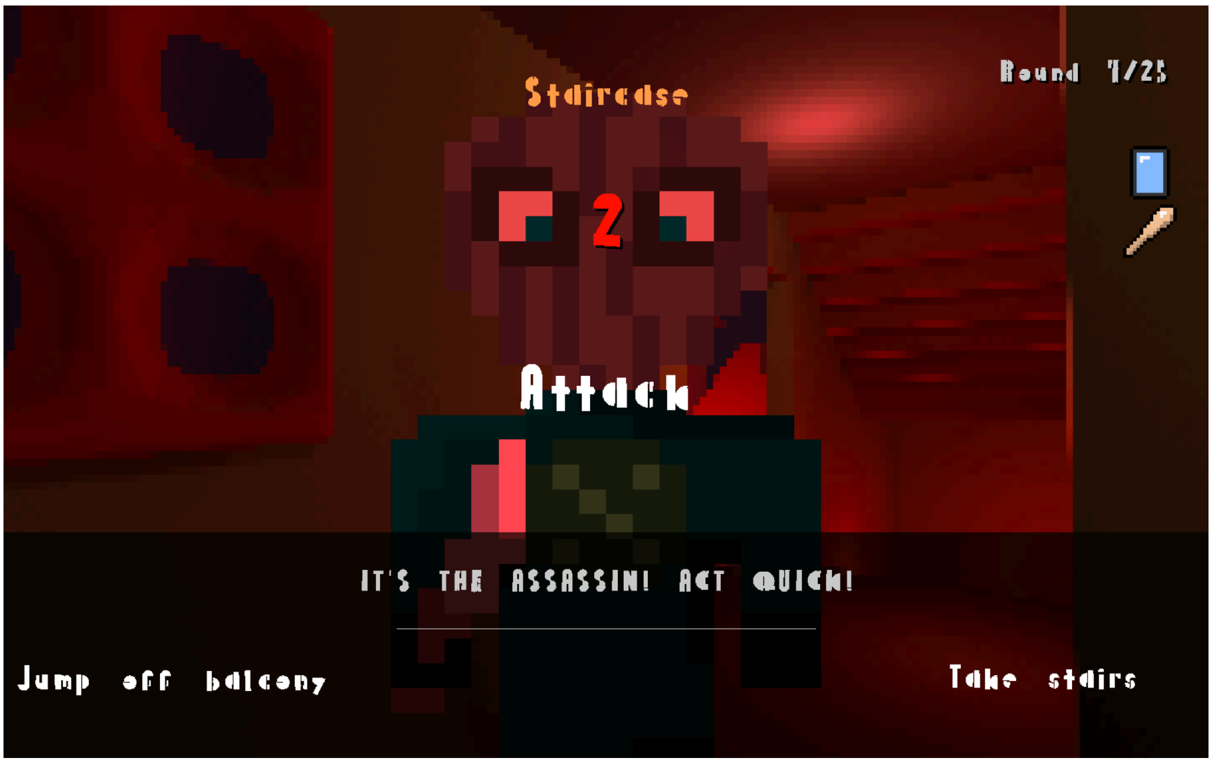
Ending

Most endings display a red background similar to the red tint that gradually covers scenes. The main way to end the game is to escape the house by getting to the front lawn and clicking the “Escape” option (referred to as “escaping”). Escaping can result in a slew of different endings depending on the actions you took beforehand. However, players can get many endings from other means than escaping, such as detonating the bomb, waiting out the fire in the bunker, or remaining in the void.

Options are clickable texts that roughly describe their function, such as “Open door” or “Grab water”. Anywhere from one to three options will be displayed along the bottom of the screen. Options can open new scenes, grant information and items, use items, and more.

Characters

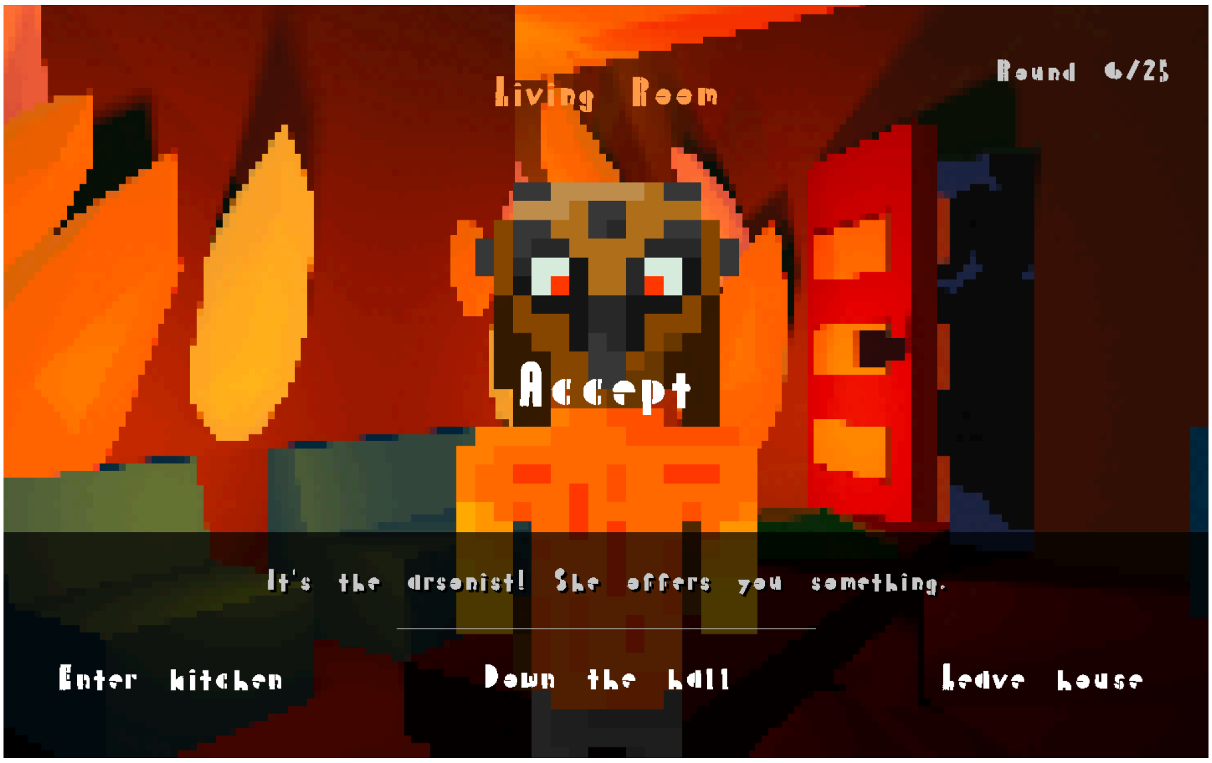
When entering a scene, players will frequently encounter characters. Characters will appear in the center of the screen, usually with an option to interact with them. Some follow a predetermined path while others follow different rules. Characters can help the player by giving or trading them items or hinder the player by attacking them.



IT'S THE ASSASSIN! ACT QUICK!

Assassin

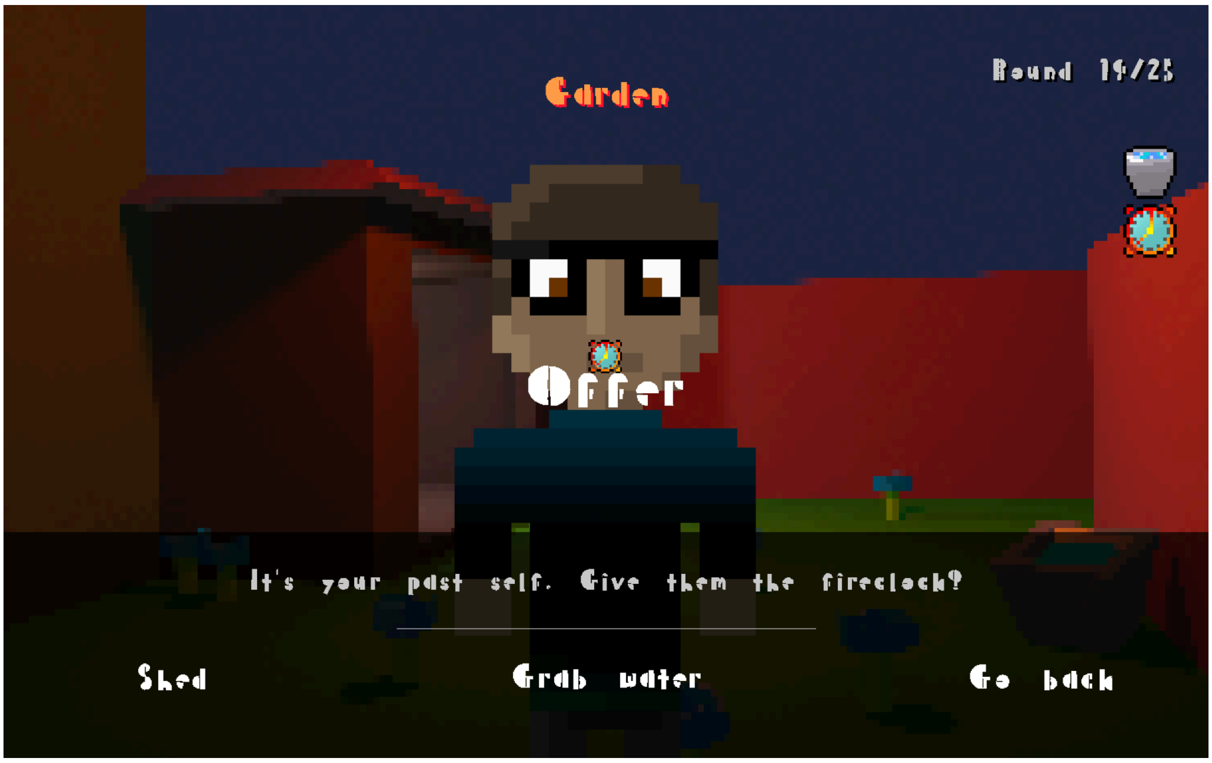
The assassin is usually the first character the player encounters. When the assassin is in a scene, he will approach the player and the timer will drop to four seconds, forcing the player to act fast. If the timer ends, the assassin will attack the player, resulting in the Assassination Ending. If the player has the bat or the daughter, an option will appear to attack the assassin. Attacking the assassin will knock him out and he will not reappear for the remainder of the game. It also allows the player to get the Revenge Ending when escaping. The assassin can be dealt with in other ways as well: using the bomb against him will result in the Ironic Ending and using the eraser against him will result in the Eraser Ending when escaping.



It's the arsonist! She offers you something.

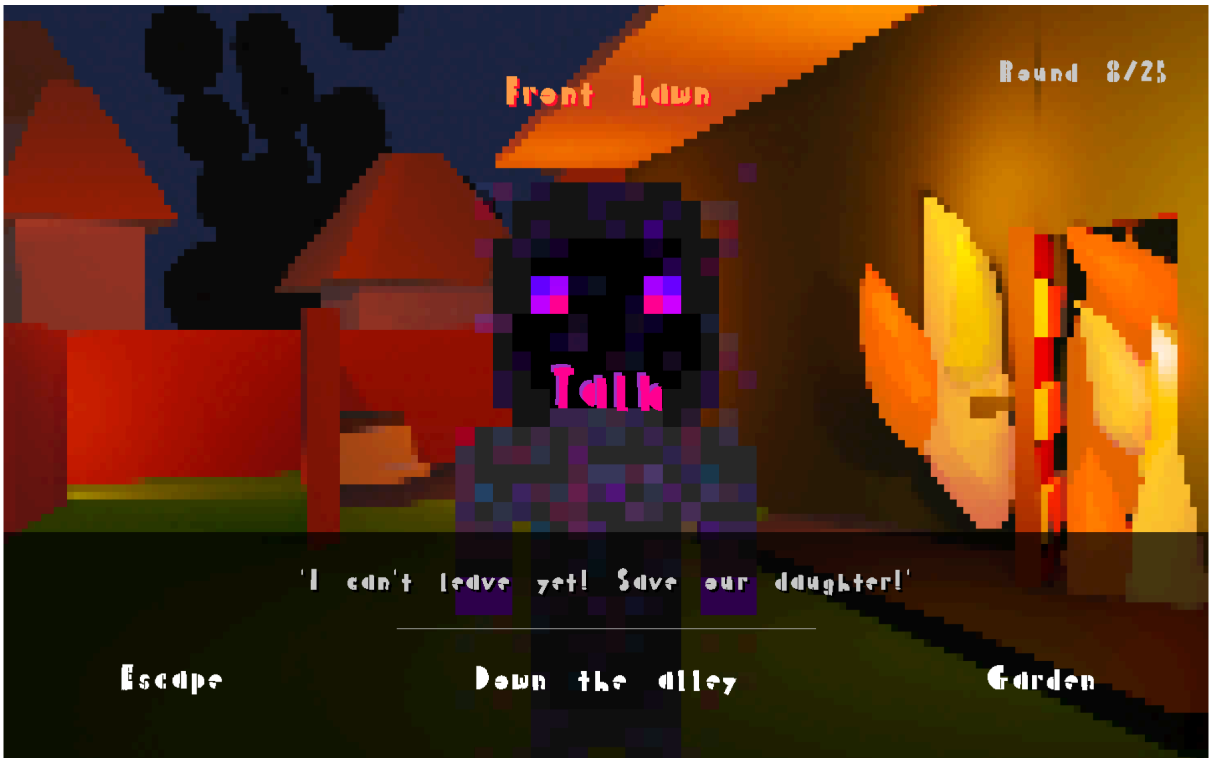
Arsonist

The arsonist is a seemingly benevolent character who offers the player items and instructions tied to the story. She moves on a set track but stays in fewer areas for longer than the assassin. The timer will disappear while the arsonist is in the scene. She helps the player complete the cross-game story, vanishing after speaking. After completing the story, she will offer some items from previous story stages that are no longer available via the previous means.



Past self

Your past self is your player character from the previous game. They only appear after retrieving the fireclock from the bunker safe. They follow the same path across scenes you took in the previous game, meaning you can move strategically in one game to make finding them easier in the next. After giving them the fireclock, they disappear from all future games.

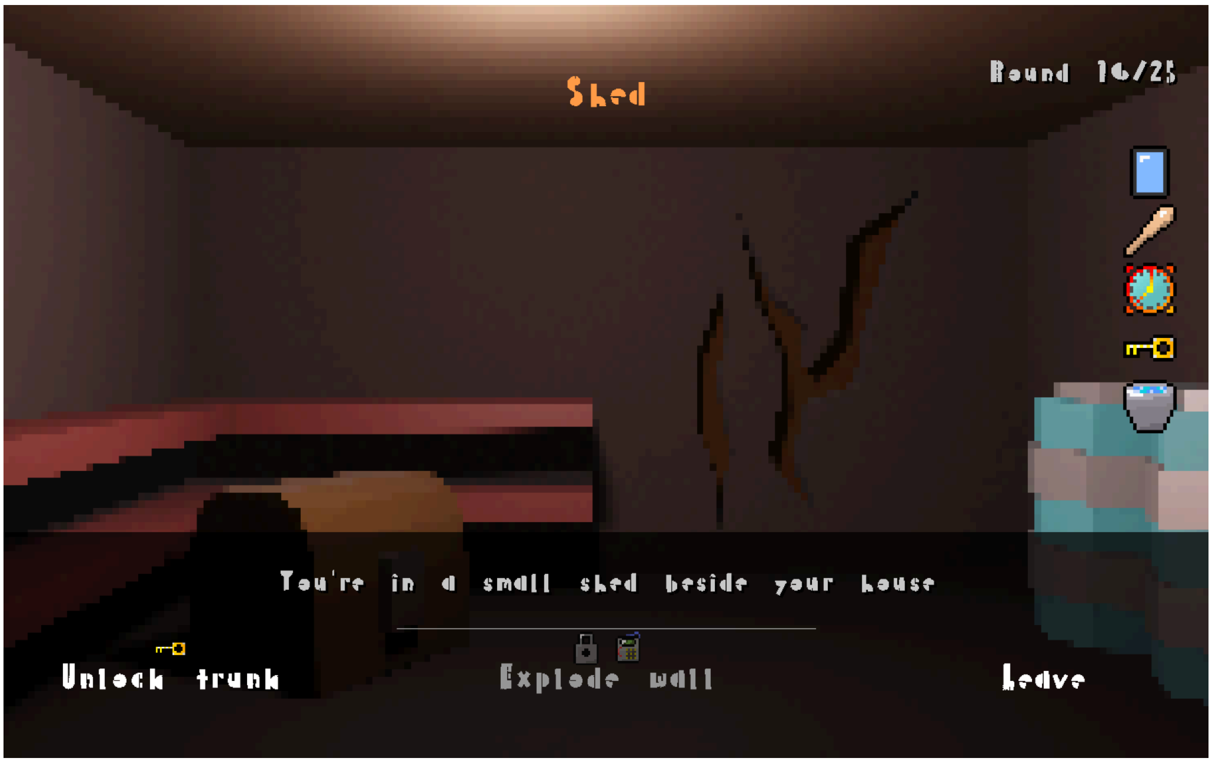


'I can't leave yet! Save our daughter!'

Partner

Your partner is the final character. They will not pick up when called on the phone until saved from the void. If you call after saving them, they will appear on the front lawn. Escaping with your daughter and partner results in the True Ending.

Items

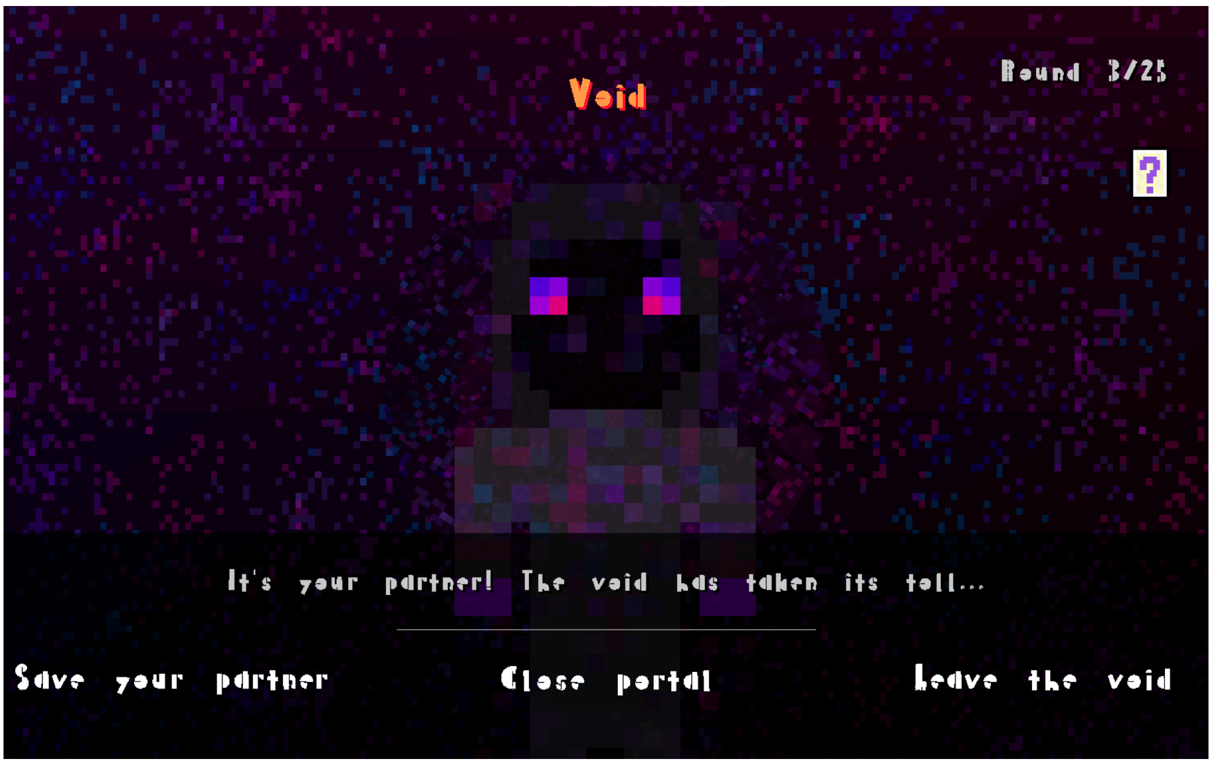


Items (displayed along the right side of the screen)

There are many items around the house that the player can collect and use. Held item icons are displayed along the right side of the screen. Items can unlock new options and scenes, among other things. Some items can be found independently of the story, while others are only accessible through the story.

Cross-game Story

Heat of the Moment features a time travel story where the player can influence future games. First, get the fireclock from the arsonist and store it in the bunker safe. Next game, you can find the fireclock in the safe and give it to your past self. You will now be able to get the eraser from the arsonist, which is used to erase the assassin from future games by sending him to the void. After erasing the assassin, the arsonist will give you the wildcard, which you will spawn with in future games. The wildcard will give you access to information about the house, a key in the closet, and a portal in the upstairs hallway. This portal will take you to the void, where you find your partner.



It's your partner! The void has taken its toll...

The Void

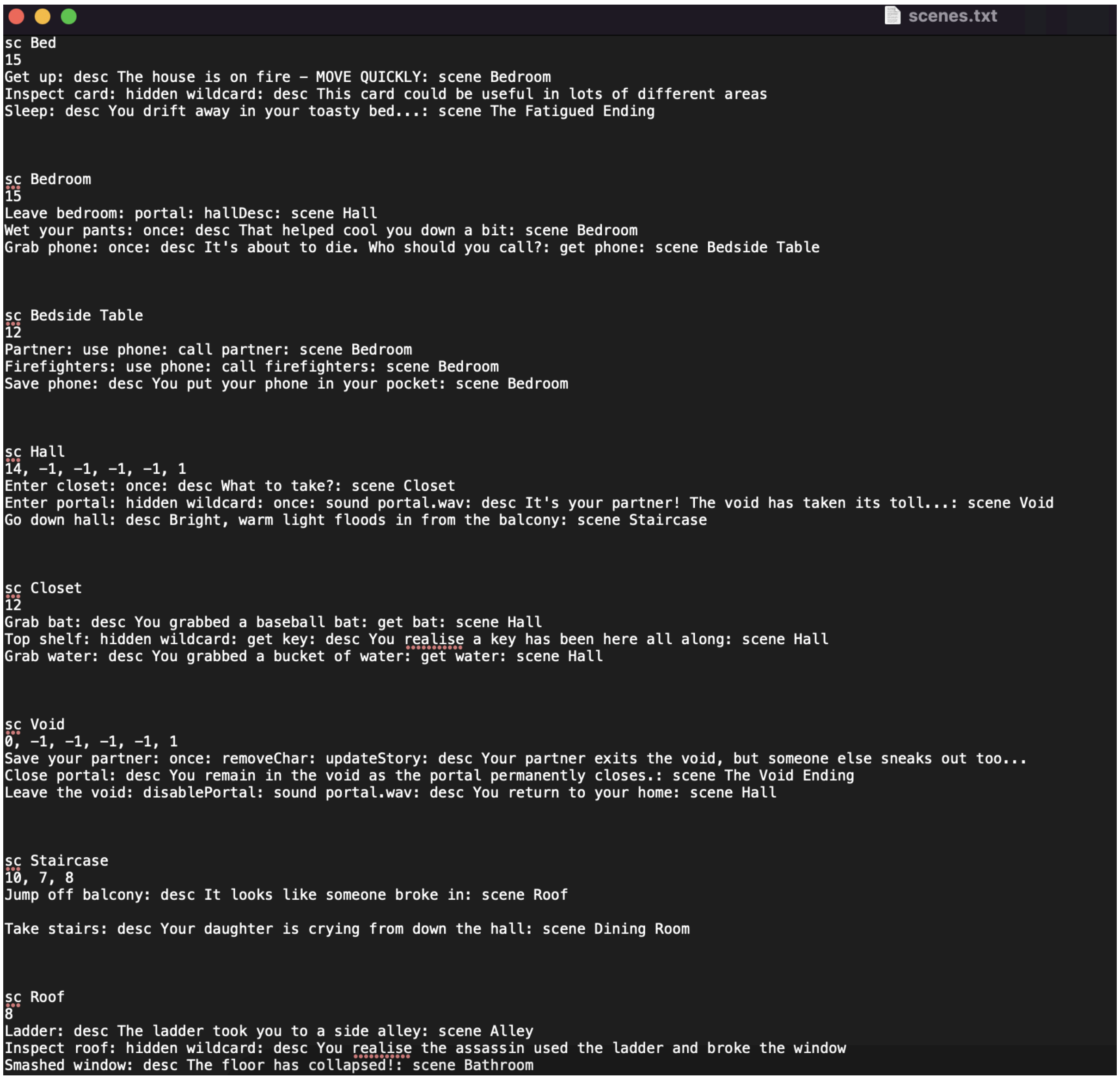
If you save your partner, the assassin will sneak out as well and continue to spawn in the house. You will no longer spawn with the wildcard, but you can now contact your partner with the phone. This is the end of the cross-game progression, but you can still access some earlier story items. You can once again retrieve the fireclock from the safe, though your past self will not appear. You can get the key from the arsonist and use it in the shed to get the elemental ring, which in turn can be traded to the arsonist for the wildcard.

Code Design

This project does not have a class hierarchy as no inheritance was necessary. The most complex bits of code are dedicated to parsing information from scenes.txt.

Scene

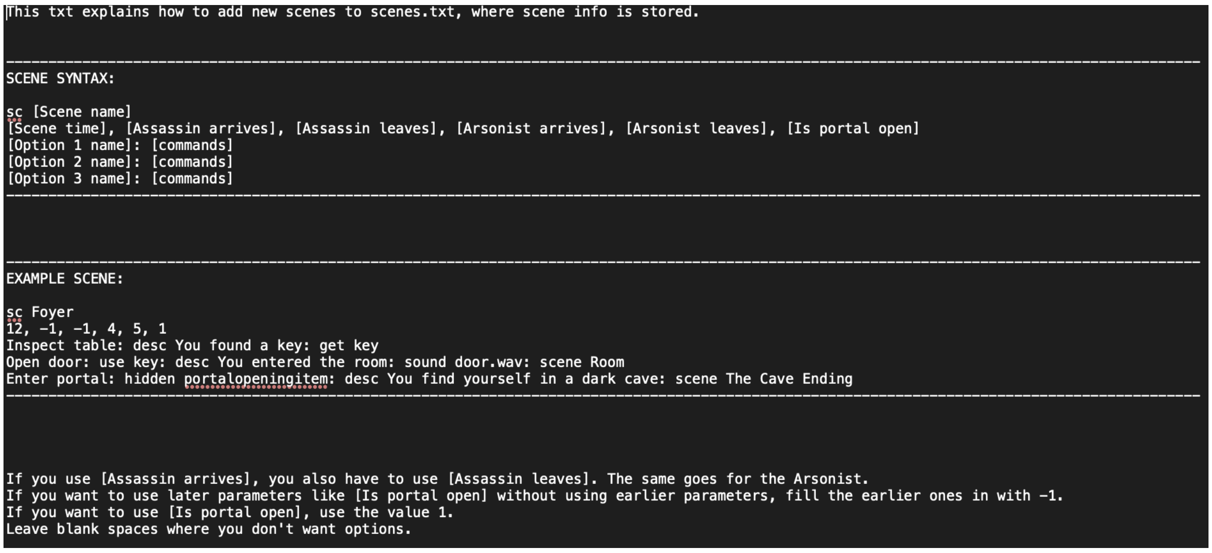
Scene is the class responsible for managing all per-scene content. It manages the background, characters, and UI. It is used for scenes, endings, and the “Play Again?” screen.



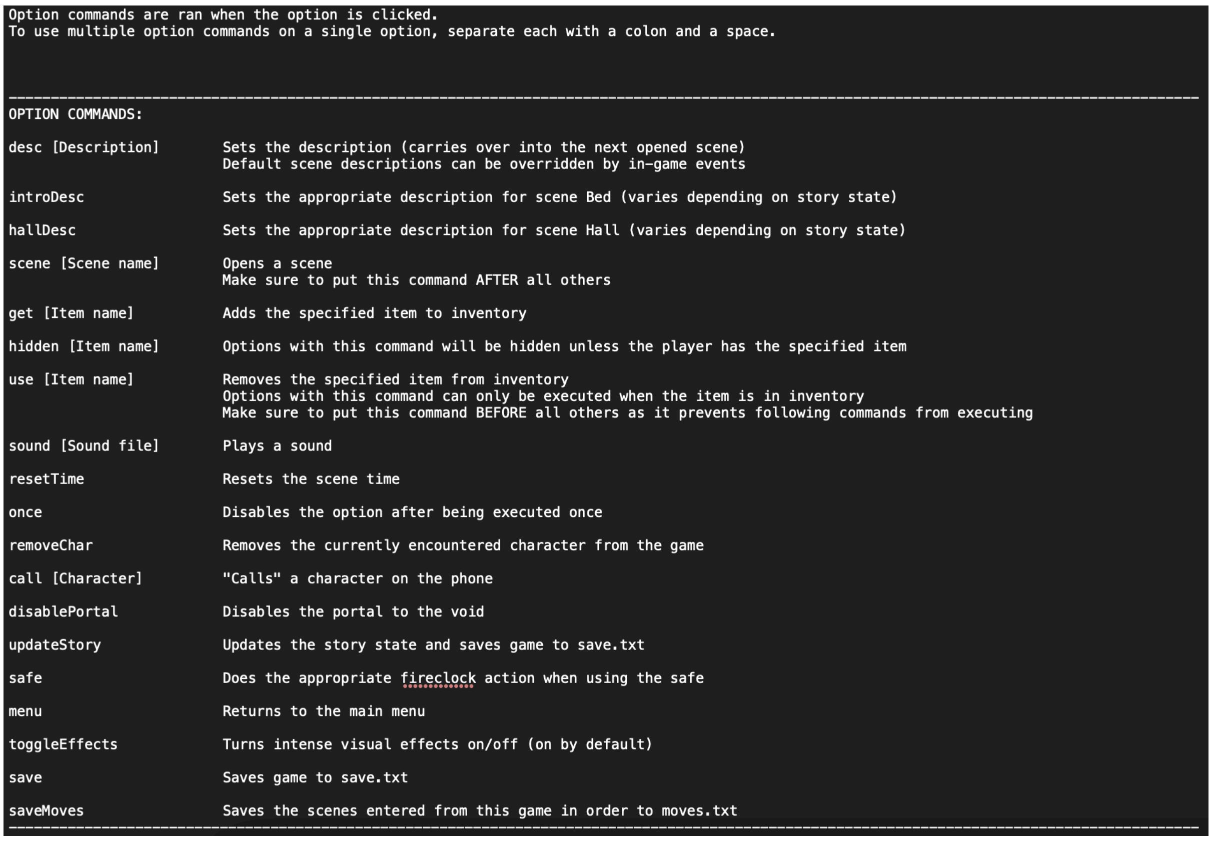
```
sc Bed
15
Get up: desc The house is on fire - MOVE QUICKLY: scene Bedroom
Inspect card: hidden wildcard: desc This card could be useful in lots of different areas
Sleep: desc You drift away in your toasty bed...: scene The Fatigued Ending


sc Bedroom
15
Leave bedroom: portal: hallDesc: scene Hall
Wet your pants: once: desc That helped cool you down a bit: scene Bedroom
Grab phone: once: desc It's about to die. Who should you call?: get phone: scene Bedside Table


sc Bedside Table
12
Partner: use phone: call partner: scene Bedroom
Firefighters: use phone: call firefighters: scene Bedroom
Save phone: desc You put your phone in your pocket: scene Bedroom


sc Hall
14, -1, -1, -1, -1, 1
Enter closet: once: desc What to take?: scene Closet
Enter portal: hidden wildcard: once: sound portal.wav: desc It's your partner! The void has taken its toll...: scene Void
Go down hall: desc Bright, warm light floods in from the balcony: scene Staircase


sc Closet
12
Grab bat: desc You grabbed a baseball bat: get bat: scene Hall
Top shelf: hidden wildcard: get key: desc You realise a key has been here all along: scene Hall
Grab water: desc You grabbed a bucket of water: get water: scene Hall


sc Void
0, -1, -1, -1, -1, 1
Save your partner: once: removeChar: updateStory: desc Your partner exits the void, but someone else sneaks out too...
Close portal: desc You remain in the void as the portal permanently closes.: scene The Void Ending
Leave the void: disablePortal: sound portal.wav: desc You return to your home: scene Hall


sc Staircase
10, 7, 8
Jump off balcony: desc It looks like someone broke in: scene Roof

Take stairs: desc Your daughter is crying from down the hall: scene Dining Room


sc Roof
8
Ladder: desc The ladder took you to a side alley: scene Alley
Inspect roof: hidden wildcard: desc You realise the assassin used the ladder and broke the window
Smashed window: desc The floor has collapsed!: scene Bathroom
```

scenes.txt

All scene information is stored in and loaded from scenes.txt. This file provides an easy way to add, edit, and delete scenes. Adding a scene is as easy as adding the scene name, parameters, and options to the file. Scenes try to open a .png of the same name to use as the background but will use the red ending background if no background can be found. Scene loading is designed to be as straightforward as possible by making most parameters optional.



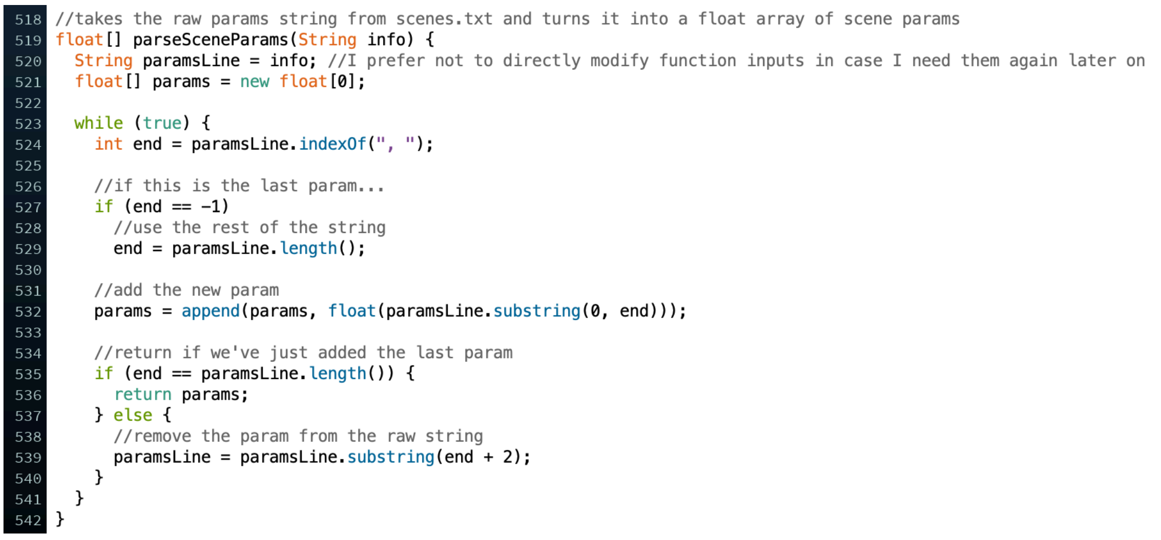
```
This txt explains how to add new scenes to scenes.txt, where scene info is stored.


--------------------------------------------------------------------------------------------------------------------------------------------------
SCENE SYNTAX:

sc [Scene name]
[Scene time], [Assassin arrives], [Assassin leaves], [Arsonist arrives], [Arsonist leaves], [Is portal open]
[Option 1 name]: [commands]
[Option 2 name]: [commands]
[Option 3 name]: [commands]
--------------------------------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------------------------------
EXAMPLE SCENE:

sc Foyer
12, -1, -1, 4, 5, 1
Inspect table: desc You found a key: get key
Open door: use key: desc You entered the room: sound door.wav: scene Room
Enter portal: hidden portalopeningitem: desc You find yourself in a dark cave: scene The Cave Ending
--------------------------------------------------------------------------------------------------------------------------------------------------




If you use [Assassin arrives], you also have to use [Assassin leaves]. The same goes for the Arsonist.
If you want to use later parameters like [Is portal open] without using earlier parameters, fill the earlier ones in with -1.
If you want to use [Is portal open], use the value 1.
Leave blank spaces where you don't want options.
```



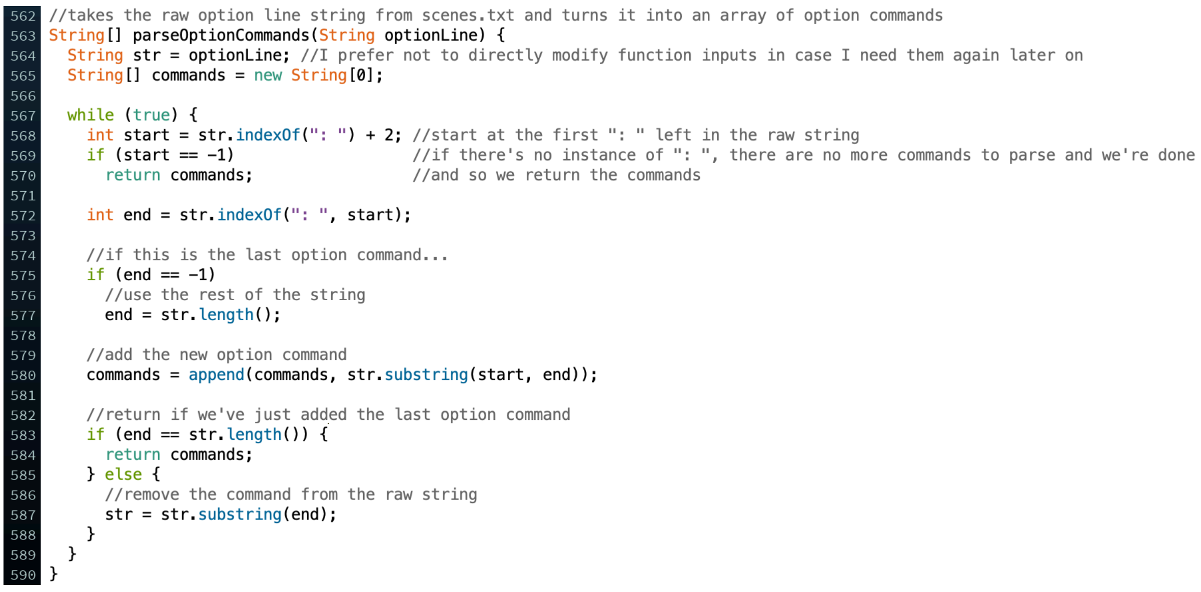
```
Option commands are ran when the option is clicked.
To use multiple option commands on a single option, separate each with a colon and a space.



--------------------------------------------------------------------------------------------------------------------------------------------------
OPTION COMMANDS:

desc [Description]         Sets the description (carries over into the next opened scene)
                           Default scene descriptions can be overridden by in-game events

introDesc                  Sets the appropriate description for scene Bed (varies depending on story state)

hallDesc                   Sets the appropriate description for scene Hall (varies depending on story state)

scene [Scene name]         Opens a scene
                           Make sure to put this command AFTER all others

get [Item name]            Adds the specified item to inventory

hidden [Item name]         Options with this command will be hidden unless the player has the specified item

use [Item name]            Removes the specified item from inventory
                           Options with this command can only be executed when the item is in inventory
                           Make sure to put this command BEFORE all others as it prevents following commands from executing

sound [Sound file]         Plays a sound

resetTime                  Resets the scene time

once                       Disables the option after being executed once

removeChar                 Removes the currently encountered character from the game

call [Character]           "Calls" a character on the phone

disablePortal              Disables the portal to the void

updateStory                Updates the story state and saves game to save.txt

safe                       Does the appropriate fireclock action when using the safe

menu                       Returns to the main menu

toggleEffects              Turns intense visual effects on/off (on by default)

save                       Saves game to save.txt

saveMoves                  Saves the scenes entered from this game in order to moves.txt
--------------------------------------------------------------------------------------------------------------------------------------------------
```

addingscenesREADME.txt explains scenes.txt syntax for proper parsing

addingscenesREADME.txt details the syntax for scenes and option commands in scenes.txt as well as a complete list of option commands. This README makes it easy to learn how to add new scenes to the game.



```
//takes the raw params string from scenes.txt and turns it into a float array of scene params
float[] parseSceneParams(String info) {
  String paramsLine = info; //I prefer not to directly modify function inputs in case I need them again later on
  float[] params = new float[0];

  while (true) {
    int end = paramsLine.indexOf(", ");

    //if this is the last param...
    if (end == -1)
      //use the rest of the string
      end = paramsLine.length();

    //add the new param
    params = append(params, float(paramsLine.substring(0, end)));

    //return if we've just added the last param
    if (end == paramsLine.length()) {
      return params;
    } else {
      //remove the param from the raw string
      paramsLine = paramsLine.substring(end + 2);
    }
  }
}
```



```
//takes the raw option line string from scenes.txt and turns it into an array of option commands
String[] parseOptionCommands(String optionLine) {
  String str = optionLine; //I prefer not to directly modify function inputs in case I need them again later on
  String[] commands = new String[0];

  while (true) {
    int start = str.indexOf(": ") + 2; //start at the first ": " left in the raw string
    if (start == -1)                   //if there's no instance of ": ", there are no more commands to parse and we're done
      return commands;                 //and so we return the commands

    int end = str.indexOf(": ", start);

    //if this is the last option command...
    if (end == -1)
      //use the rest of the string
      end = str.length();

    //add the new option command
    commands = append(commands, str.substring(start, end));

    //return if we've just added the last option command
    if (end == str.length()) {
      return commands;
    } else {
      //remove the command from the raw string
      str = str.substring(end);
    }
  }
}
```

These functions parse raw scene information into arrays

Whenever a scene is opened, scenes.txt is loaded and relevant scene information is taken from it. Scene parameters are parsed into a float array with parseSceneParams() and passed to the new Scene. parseOptionName() and parseOptionCommands() parse each option line into an option name and an array of option command Strings, all of which is passed on to a TextBox through the Scene. The option commands will be fully parsed later when the option is clicked.

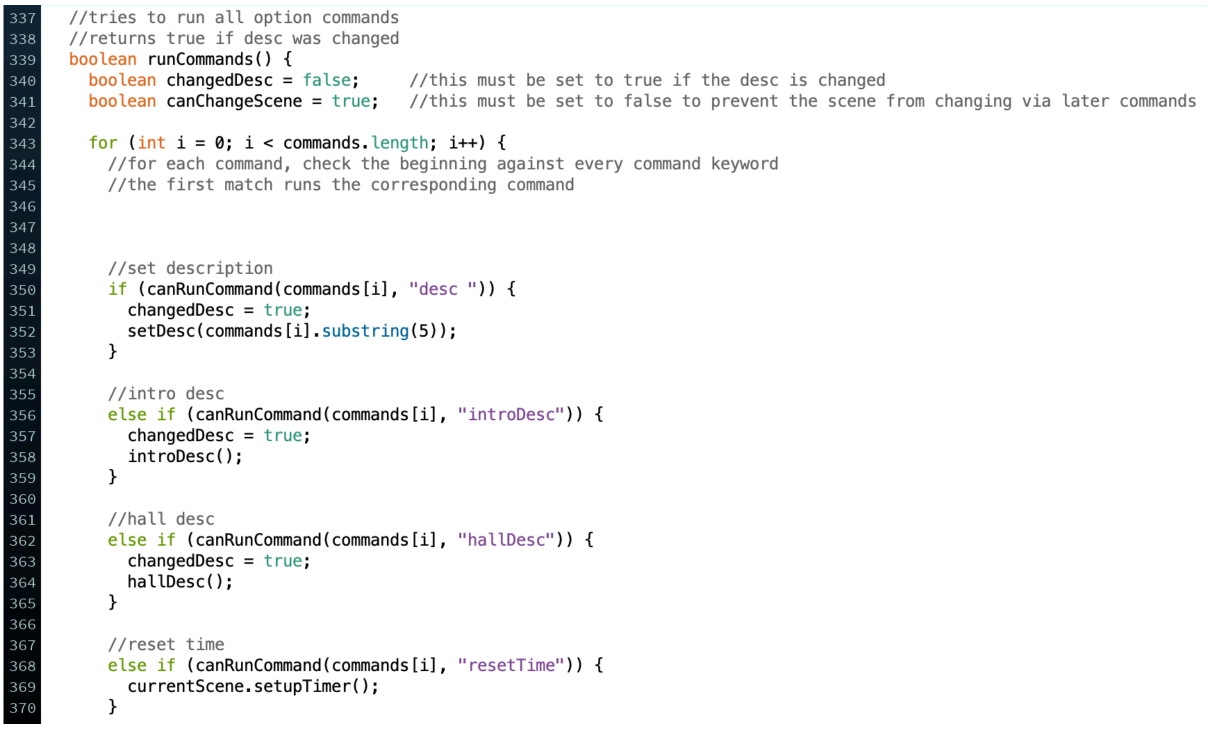
Scene parameters are as follows: timer duration in seconds, the round when the assassin enters the scene, the round when the assassin leaves the scene, the round when the arsonist enters the scene, the round when the arsonist leaves the scene, and if a portal is open in the scene. All parameters other than the timer duration are optional and can be left blank. The timer can be disabled by setting the timer duration to 0.

TextBox

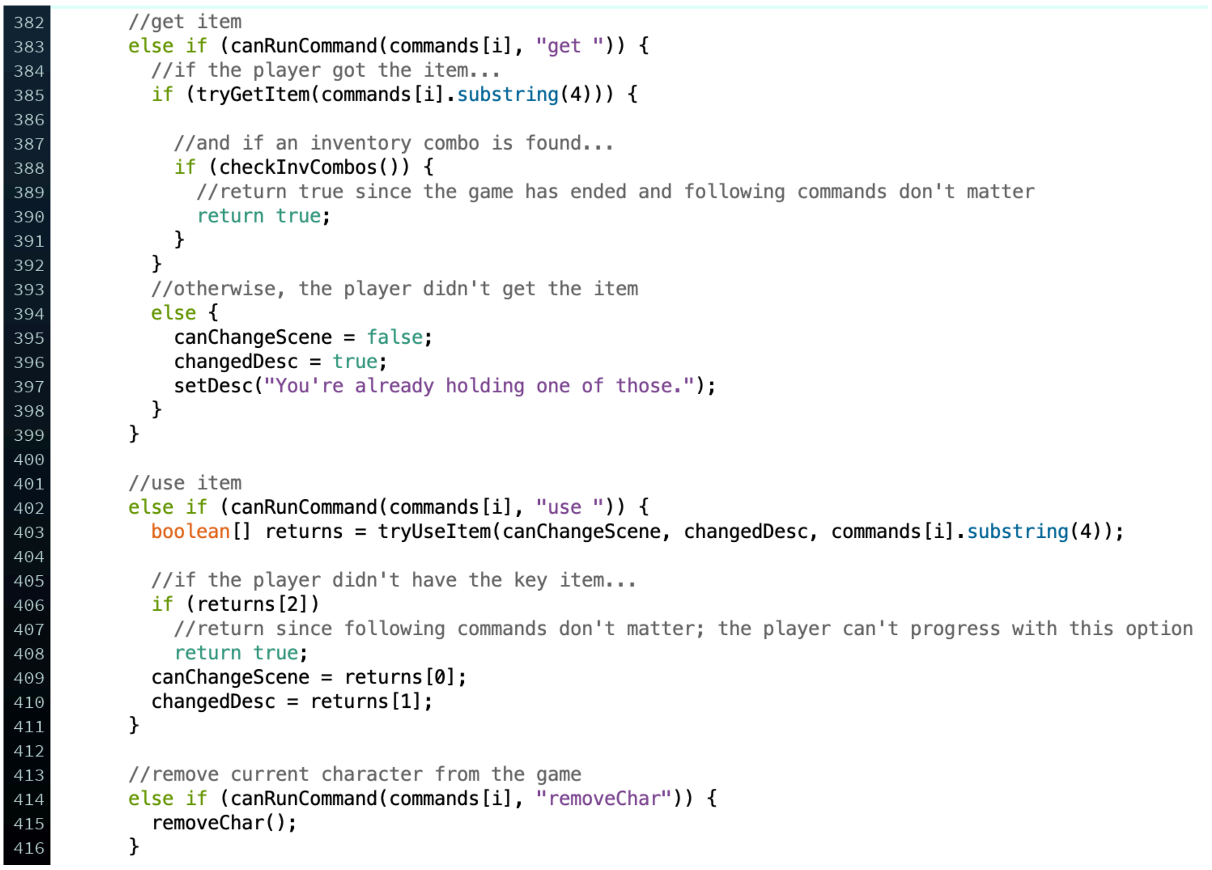
The TextBox class is used for all in-game text. TextBox’s can be clickable, disabled, or hidden. They have modifiable hitboxes, colours, sizes, and shake amounts. When hovered, a TextBox will show a secondary flickering “shadow” text effect of a specified colour and shake harder. Title TextBox’s will always display this effect under the original text.

Clickable TextBox’s are referred to as “options”. In-game options displayed along the bottom of the screen are referred to as “default options”, while options that appear with characters are referred to as “character options”. An option’s functionality is determined by its option commands.

Option Commands



```
//tries to run all option commands
//returns true if desc was changed
boolean runCommands() {
  boolean changedDesc = false;     //this must be set to true if the desc is changed
  boolean canChangeScene = true;   //this must be set to false to prevent the scene from changing via later commands

  for (int i = 0; i < commands.length; i++) {
    //for each command, check the beginning against every command keyword
    //the first match runs the corresponding command



    //set description
    if (canRunCommand(commands[i], "desc ")) {
      changedDesc = true;
      setDesc(commands[i].substring(5));
    }

    //intro desc
    else if (canRunCommand(commands[i], "introDesc")) {
      changedDesc = true;
      introDesc();
    }

    //hall desc
    else if (canRunCommand(commands[i], "hallDesc")) {
      changedDesc = true;
      hallDesc();
    }

    //reset time
    else if (canRunCommand(commands[i], "resetTime")) {
      currentScene.setupTimer();
    }
```



```
    //get item
    else if (canRunCommand(commands[i], "get ")) {
      //if the player got the item...
      if (tryGetItem(commands[i].substring(4))) {

        //and if an inventory combo is found...
        if (checkInvCombos()) {
          //return true since the game has ended and following commands don't matter
          return true;
        }
      }
      //otherwise, the player didn't get the item
      else {
        canChangeScene = false;
        changedDesc = true;
        setDesc("You're already holding one of those.");
      }
    }

    //use item
    else if (canRunCommand(commands[i], "use ")) {
      boolean[] returns = tryUseItem(canChangeScene, changedDesc, commands[i].substring(4));

      //if the player didn't have the key item...
      if (returns[2])
        //return since following commands don't matter; the player can't progress with this option
        return true;
      canChangeScene = returns[0];
      changedDesc = returns[1];
    }

    //remove current character from the game
    else if (canRunCommand(commands[i], "removeChar")) {
      removeChar();
    }
```

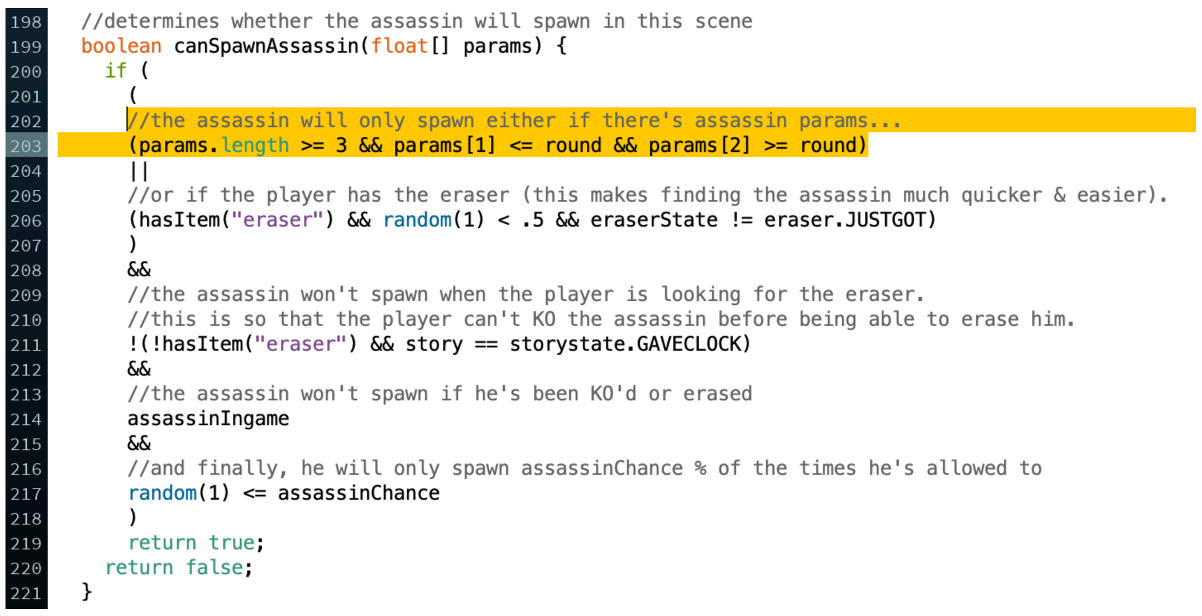
Some of the option commands parsed in runCommands()

Option commands are functions that run when an option is clicked. Some, such as “use ” and “hidden ”, affect the appearance of the option by locking, hiding, or revealing it. They are separated with a colon and a space in scenes.txt. Hardcoded option commands are formatted as though they were already parsed: as a String array without a colon and a space between each command.

Parsed option commands are stored in a String array that is iterated through in runCommands(). Each iteration will attempt to find a valid command keyword at the beginning of the String such as “sound ”. The first match it finds will run the appropriate code.

For example, assume the player clicks an option with the command “get water”. The code will check against all keywords until it finds the keyword “get ”. It will then run tryGetItem() with the input itemName being “water” – the remainder of the “get water” String when the keyword is removed. Finally, tryGetItem() will add the water Item to the inventory if one is not already there.

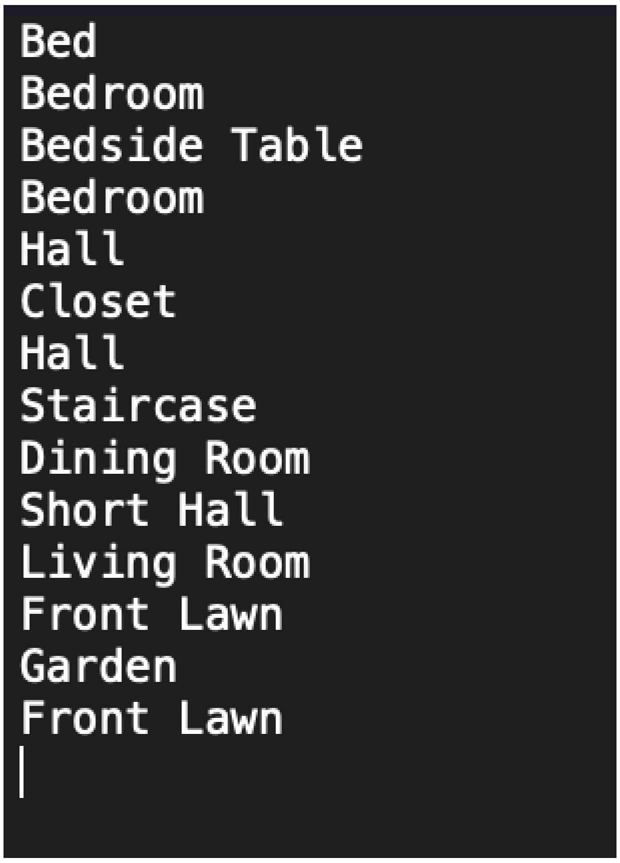
Characters



```
//determines whether the assassin will spawn in this scene
boolean canSpawnAssassin(float[] params) {
  if (
    (
    //the assassin will only spawn either if there's assassin params...
    (params.length >= 3 && params[1] <= round && params[2] >= round)
    ||
    //or if the player has the eraser (this makes finding the assassin much quicker & easier).
    (hasItem("eraser") && random(1) < .5 && eraserState != eraser.JUSTGOT)
    )
    &&
    //the assassin won't spawn when the player is looking for the eraser.
    //this is so that the player can't KO the assassin before being able to erase him.
    !(!hasItem("eraser") && story == storystate.GAVECLOCK)
    &&
    //the assassin won't spawn if he's been KO'd or erased
    assassinIngame
    &&
    //and finally, he will only spawn assassinChance % of the times he's allowed to
    random(1) <= assassinChance
    )
    return true;
  return false;
}
```

canSpawnAssassin() checks his scene parameters to see if he can spawn

Most characters appear at predetermined places and times, usually moving between adjacent areas in a realistic movement-like manner. The assassin and arsonist do this by checking their respective scene parameters. Sometimes, characters will appear in multiple scenes in the same round for increased chances of encountering them. Characters' locations update every round. The only exception to this rule is the partner, who only appears in certain scenes when story conditions are met. The player will only encounter a given character in a scene if that character's round and scene conditions are met.



```
Bed
Bedroom
Bedside Table
Bedroom
Hall
Closet
Hall
Staircase
Dining Room
Short Hall
Living Room
Front Lawn
Garden
Front Lawn
```

A game’s scene progression saved to moves.txt

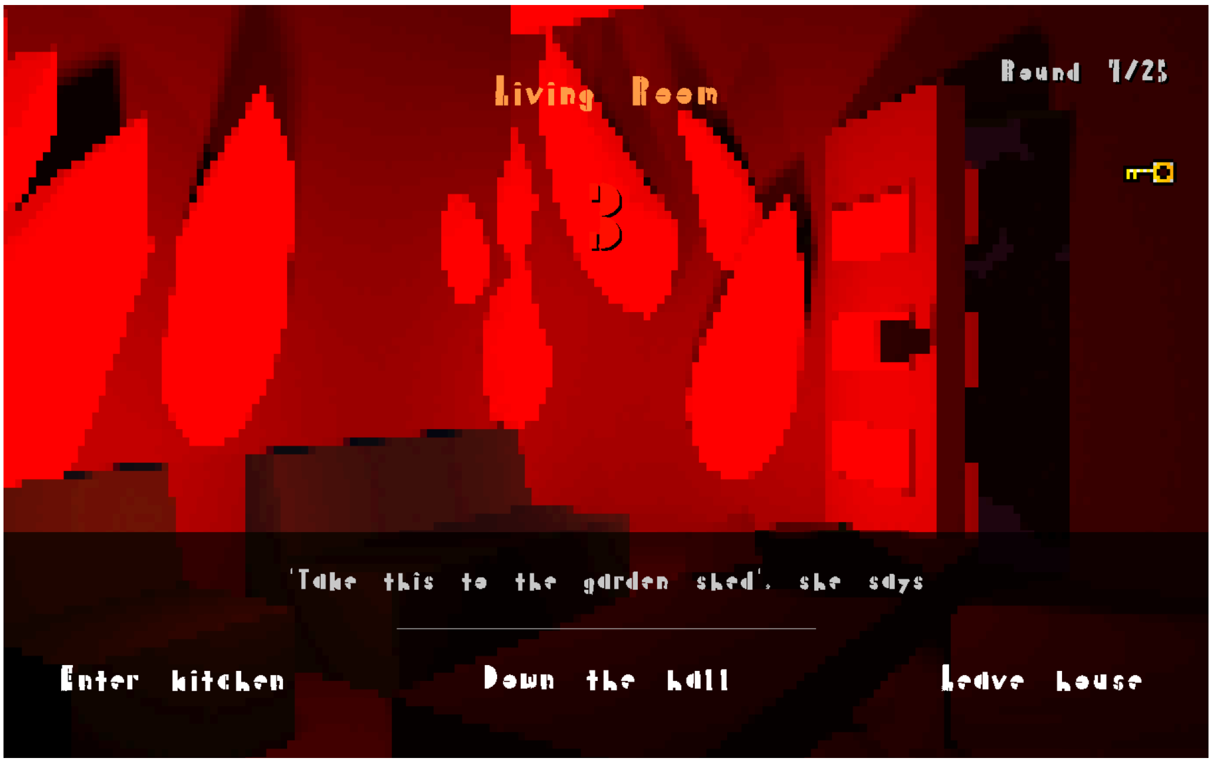
The past self’s path is not predetermined. Every time a round passes, the player’s current scene is saved to moves[]. At the end of the game, moves[] is saved to moves.txt. Next game, moves.txt will be loaded to determine where the past self is each round. The “Wait longer” option in the bunker was designed to help players spend many rounds in one scene; this makes finding the past self much easier, especially since the bunker is a key location in relation to the past self.

Reflection

Issues

I am only aware of one bug. After playing for a few minutes, sound begins distorting and clipping. I suspect this has to do with the Processing sound library and looping sounds. To my knowledge, any other issues could only be caused by syntax errors with new scenes.txt additions.

However, there are some functional elements of the game that could be improved. When entering the daughter’s room through the trapdoor in the elemental’s hideout, the player will usually have the water item due to it sitting along the path to the elemental’s hideout and having no other usage along the way. This results in the player entering the daughter’s room and immediately getting the Soaked Ending. This is because they got the daughter upon entering the daughter’s room and had the water and the daughter in their inventory.



The timer gets harder to see as the scene turns red

Additionally, the timer becomes hard to see in dense, warm-coloured areas, especially as time runs out and the scene turns red. I tried fixing this with a background behind the timer and giving the timer a different colour, but both solutions affected the visual style of the game. The font can also be a bit hard to read at times, but changing it also diminished the visual style. Character options could also be difficult to notice for new players, which I attempted to remedy by increasing their size.

In code, characters should have a dedicated class. This would clean up the mess of character functions and switch statements on Scene. There are also some rare-use option commands that could operate more cleanly had I designed the system differently.

Goals

I successfully included every plot point and gameplay feature I set out to. The cross-game story was not vital to the core gameplay, but added a greater goal to the game and boosted replayability.

The gameplay follows the curve I intended. Initially, the player is rushing through scenes as fast as possible. The time limit, intense visuals, and assassin encounters put the player in a high-stress mindset. As the player gets familiar with the game, they learn where to go and how to deal with different situations. The player gets more accustomed to the intensity of the game over time. Their focus shifts from survival to fast-paced exploration, dipping in and out of the house to transport items and seek out characters. Eventually, the game becomes about transcending the house both figuratively and literally through the cross-game progression.

The story is borderline nonsensical, especially due to the outlandish cross-game narrative. It was adapted many times to ensure it felt in place with the rest of the game and avoided ludonarrative dissonance. Though key story details can be discovered with the wildcard and inferred through environmental storytelling and scene information, connecting most of the dots is left to the player.

I am very pleased with the visual aesthetic of the game. I rendered 3D scenes in Blender and pixelated them to fit with the pixel art I made in Aseprite, which worked exactly as I envisioned. Though some scenes are a bit busy, it is normally due to the scene being aflame. I believe this adds to the overwhelming nature of these scenes in combination with their short time limit. The shaking of the background and UI, alongside the effects that gradually intensify over time, effectively communicate the intensity and danger of being inside a burning house.

I believe the sound design could be improved, but it does not currently detract from the experience in my opinion. I made all sound effects in Logic Pro, mostly with the Massive synth. It proved tricky to create ambient fire noise with a synth, resulting in a strange white noise behind most areas of the game. I decided to keep the noise rather than use a stock fire sound as I wanted to create all game assets myself.

Word count: 2698