May 9, 2022

Choose Your Own Adventure: Labors of Hercules

Spencer Lyudovyk, Amanda Lin, Jue Gong
HTHS ✥ CSE ✥ AG22_6

Project Description

The program in its entirety culminates in a game, where the player becomes a (gender-neutral) demigod in their early youth, eager to prove themself. Stuck within a labyrinth, the player awaits imminent assaults from half a dozen monsters, and needs to defeat them all in order to be bestowed with the elixir of immortality. The player can move around the labyrinth and enter different rooms scattered throughout the labyrinth's perimeter. User inputs can dictate a variety of tasks, such as:

(1) picking up weapons
(2) retaliating the monster’s attacks
(3) opening locked doors
(4) lighting up dark regions

Program Requirements

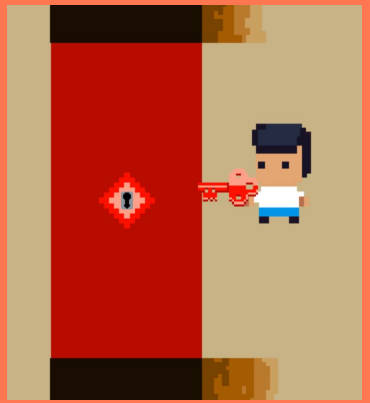
1. The entire game must be programmed with Python 3.
2. The game should be implemented with a well-designed and easy-to-use graphical user interface.
3. A minimum of 6 rooms (or something equivalent to rooms). Each team member is responsible for designing two rooms.
4. At least one “locked” door that can only be entered if the user finds an item to open the door.

1

2

3 (and classes)

4



Program Requirements

5. At least one room that is too dark to see into without first finding a light source.
6. At least one room that can only be opened if a certain number of items has been found.
7. At least one list variable that keeps track of all items found by the user.

5

The middle of the labyrinth has to be lit by a light source

6

Must collect at least 5 gems to open the last door

7

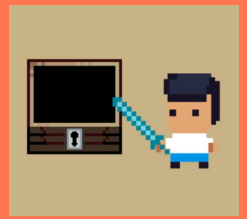
```
items_list = []
```

The “backpack” that contains the items picked up by the player

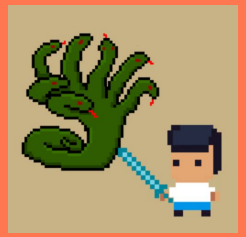
Program Requirements

8. The user should be able to interact with objects in the room.
   a. Ex: open drawers in furniture, pick up objects, look under objects
9. To carry items, the player has a backpack, which, when accessed, displays the contents (including quantities) in the form of a list. The backpack must have a limit on the number of items (it doesn't provide infinite space). If you like you may also include encumbrance rules for determining how much the player's equipment slows them down.

8

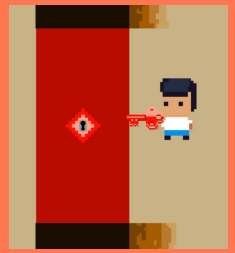


Taking things out of the chest



Attacking the monster

Dig up plants/apples



Open doors

9

Max: 5 things in hand and backpack combined

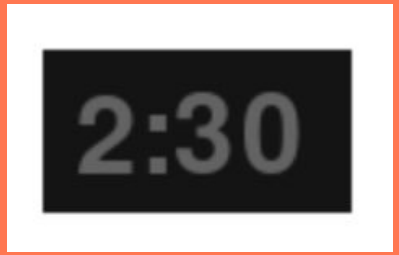
Program Requirements

10. The user will be prompted if trying to perform an action that is impossible
    a. Ex: trying to open a door that is locked, trying to walk in a direction out of a room where there is no door, trying to pick up an object that is too large, trying to open an object that can't open, trying to perform an action that isn't defined, etc.
    b. A room in which something happens if the user is holding a specific object.
11. (Optional) To increase the player challenge consider:
    a. Implementing a timer so that the player has to complete the game in a certain time.
    b. Implement encumbrance (carrying capacity) rules that determine how much the player is slowed down by their equipment.
    c. Have the player be chased through the game by a creature (think Stranger Things or IT)

10

You can't kill with the key.

11



Program Features Set/Requirements

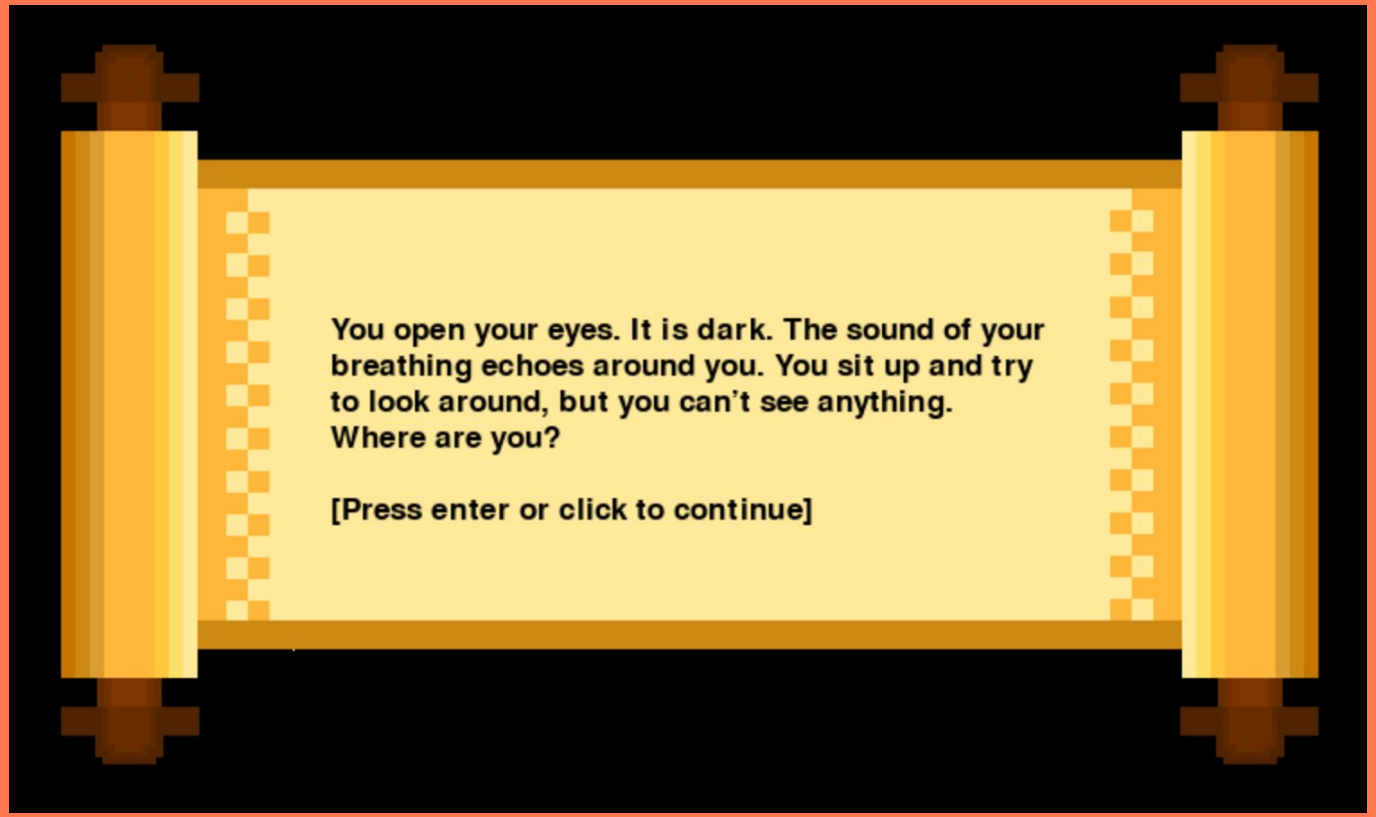
- Classes are used as room equivalents, including:
  - Background, Door, Player, Monster (6 subclasses for 6 different monsters), Holdable (each weapon/item that the player can hold is a subclass), Weapon (each item that can do damage is a weapon), etc
- Locked doors:
  - The door to each monster's room can only be opened with the key of the same color as the door
- Room that is too dark to see:
  - Labyrinth starts off fully dark
  - One square in the labyrinth (the square the player is in) lights up if the player is holding a turned on flashlight
  - Entire labyrinth lights up if the player clicks the light switch in the middle of the labyrinth
- Room requiring certain number of items:
  - Final room only opens once the player has collected and placed 5 gems in front of the door
- At least one list variable that keeps track of all items found by the user:
  - available_weapons keeps track of the items a player can pick up in the labyrinth
  - items_list keeps track of the items the player is holding
- Object interaction:
  - Digging up plants with shovel for increased health
  - Unlocking doors
  - Opening chests and taking objects out of them
  - Picking up/dropping/adding to backpack any holdable items in the labyrinth
  - Killing monsters

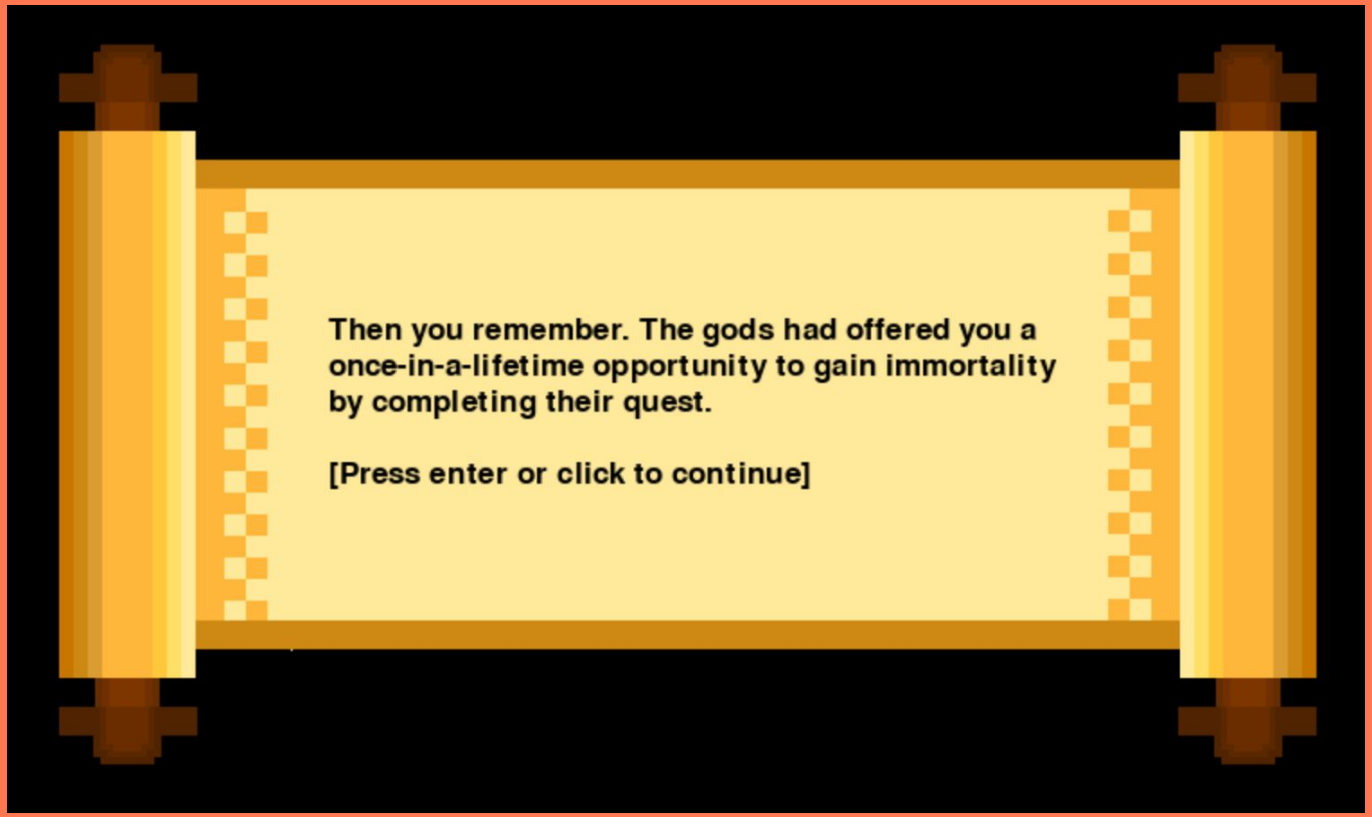
Program Features Set/Requirements

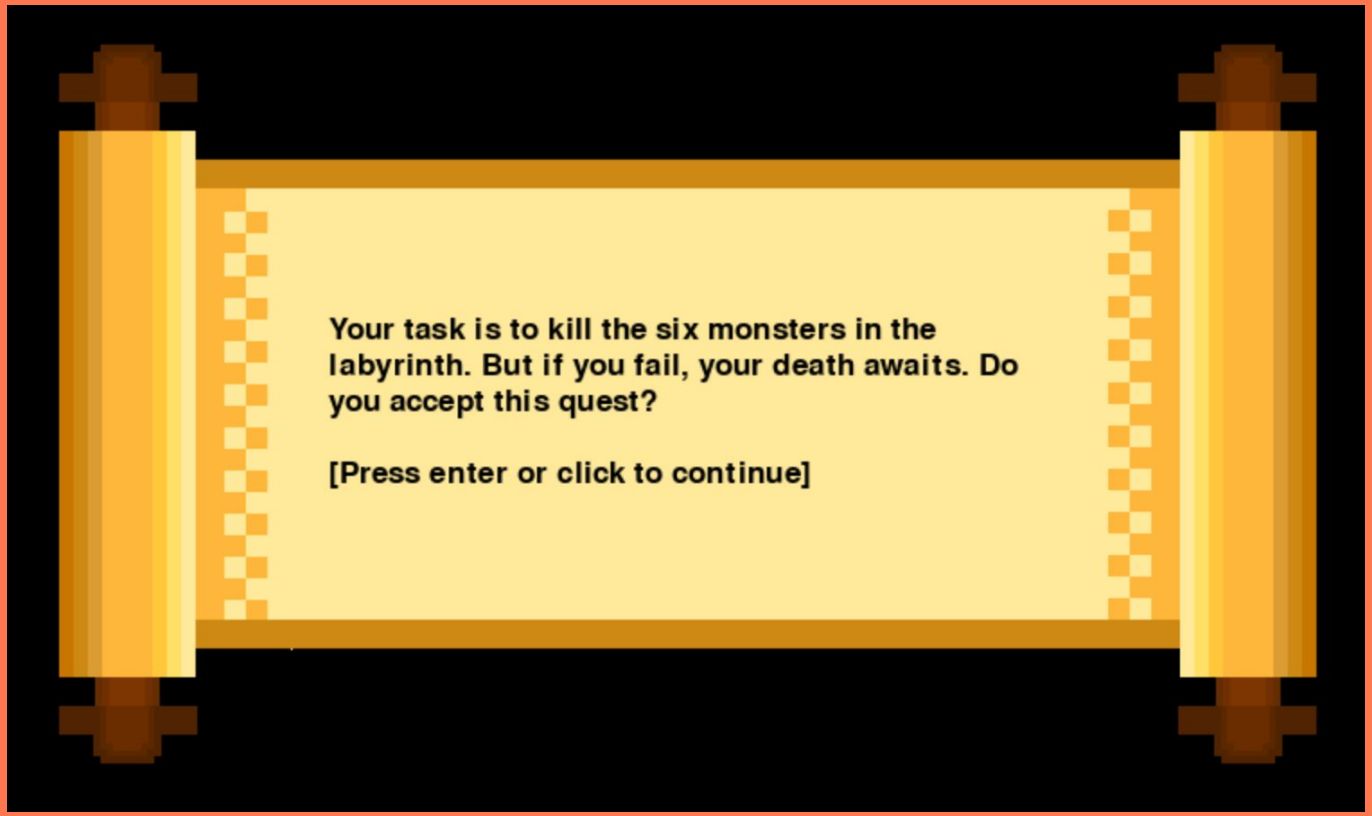
- Storyline is implemented through Story class that gives background information and instructions
- Player backpack:
  - Player can carry a maximum of 5 items combined in their backpack and in their hands
  - Player can access a menu that displays a list of all the items they are holding
  - Player can click on each item in the menu to view how many damage points each item can do to each monster
  - User can click on the number corresponding to an item in the menu to select it from the backpack
  - Item quantities in the menu are all 1 to allow for selection of exact items (even if items are similar (e.g., keys))
- Prompts for impossible actions:
  - A red error message is displayed at the bottom of the screen for any actions the player shouldn't or can't take
  - They include: using the wrong color key on a door; not having enough gems to open the last room; trying to dig with an item other than a shovel; trying to kill with an item that isn't a weapon; trying to drop a weapon while not holding anything; trying to pick up a weapon while already holding another weapon; trying to pick up more than 5 items; etc
- A room in which something happens if the user is holding a specific object:
  - Picking up potion in the final room gives the player immortality
  - Placing items (e.g., gems) into the space in front of the last room results in attempt to unlock the final door
  - Labyrinth partially lights up around player if holding flashlight
  - "First encounter" instructions (e.g., informs the user what to do when they encounter a plant for the first time)
- Timer: gems disappear after 8 minutes if not collected
- Player being chased: each of the 6 monsters is able to track and attack/kill the player
- If you die while playing or don't collect enough gems, you have the option to replay the game

Background Story

You wake up as a demigod...


You open your eyes. It is dark. The sound of your breathing echoes around you. You sit up and try to look around, but you can't see anything. Where are you?
[Press enter or click to continue]

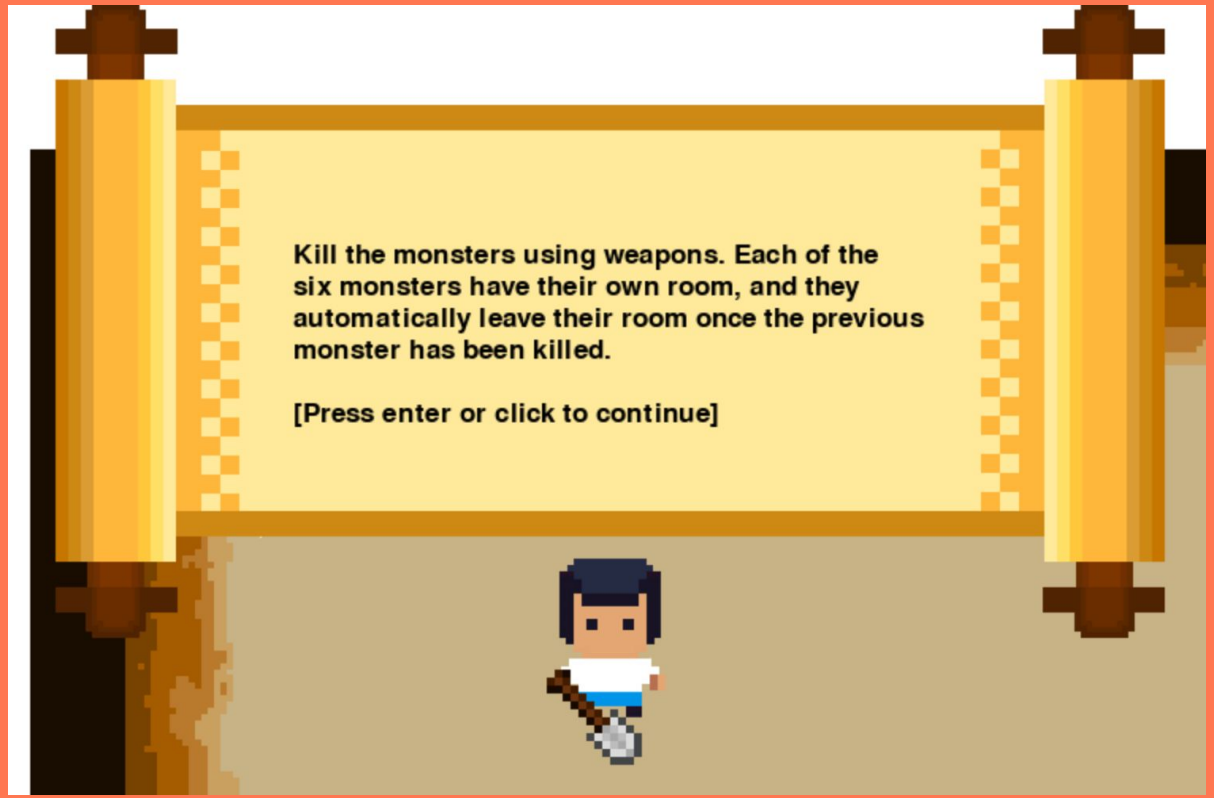

Then you remember. The gods had offered you a once-in-a-lifetime opportunity to gain immortality by completing their quest.
[Press enter or click to continue]

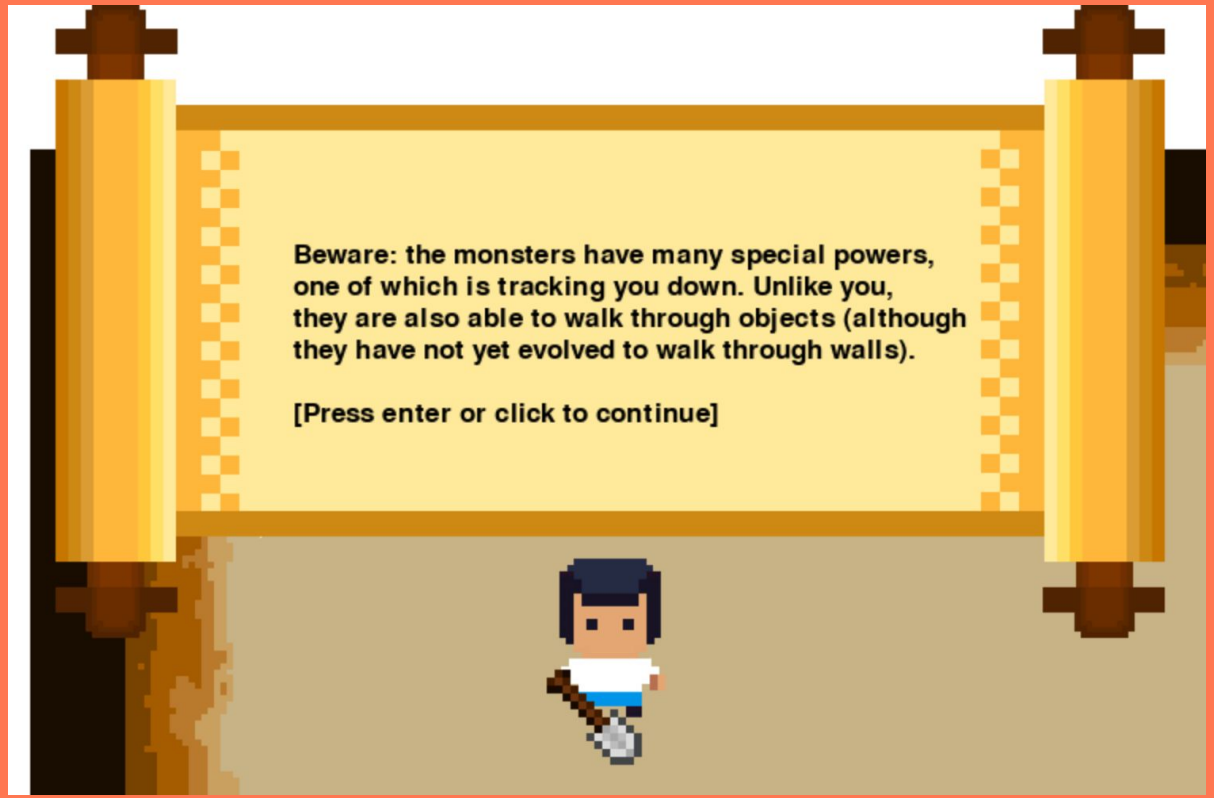

Your task is to kill the six monsters in the labyrinth. But if you fail, your death awaits. Do you accept this quest?
[Press enter or click to continue]

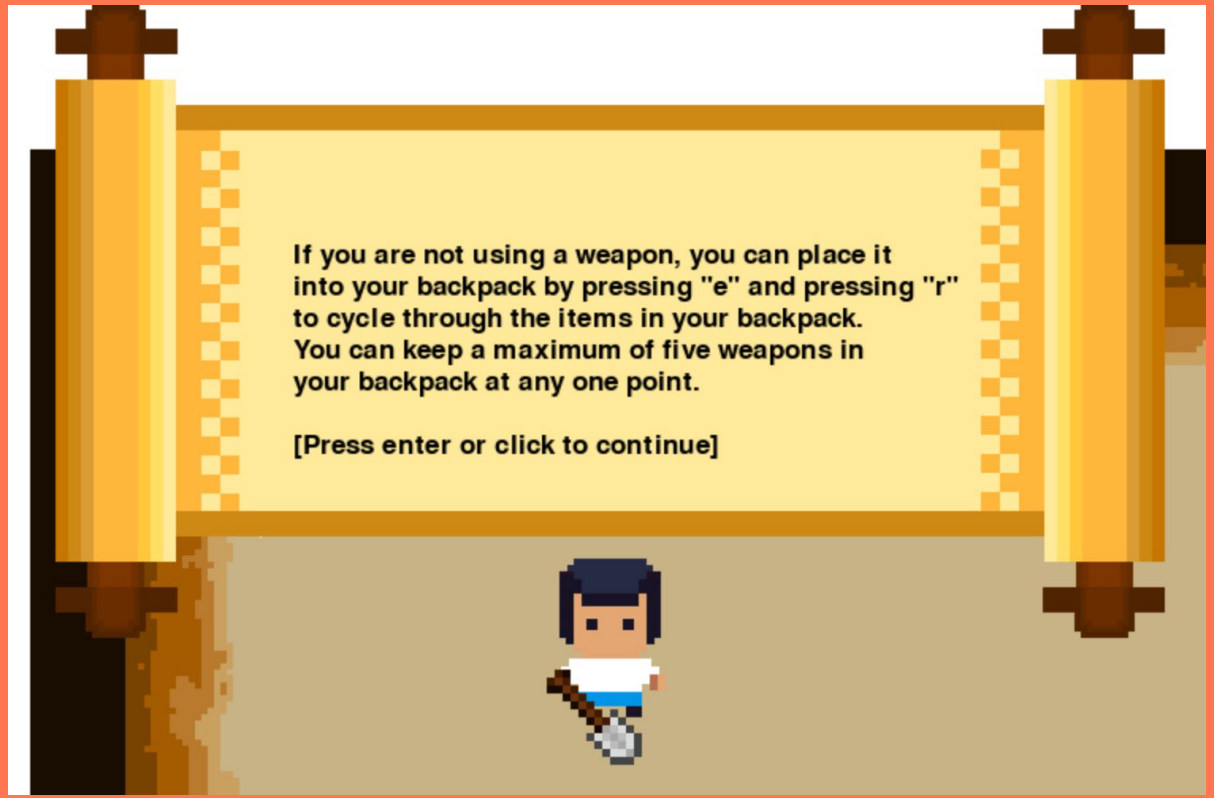
Instructions

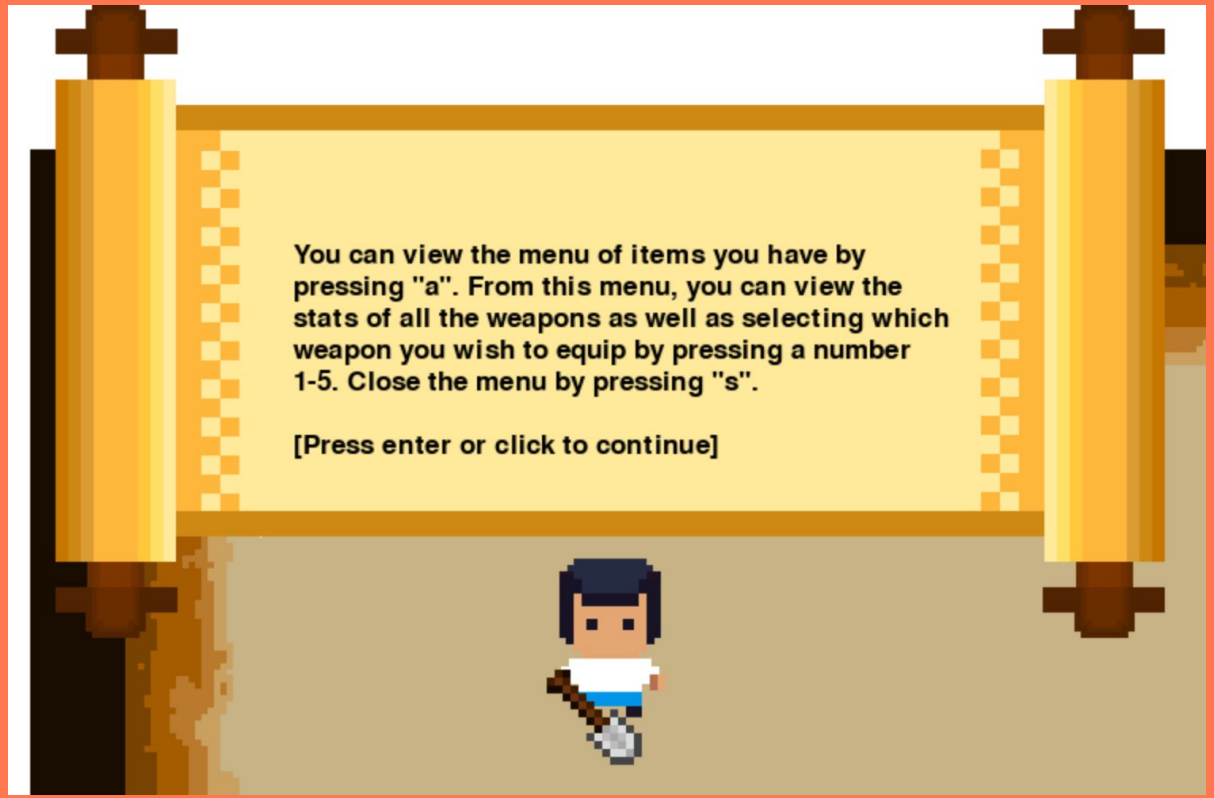
The ultimate guide to achieve immortality?

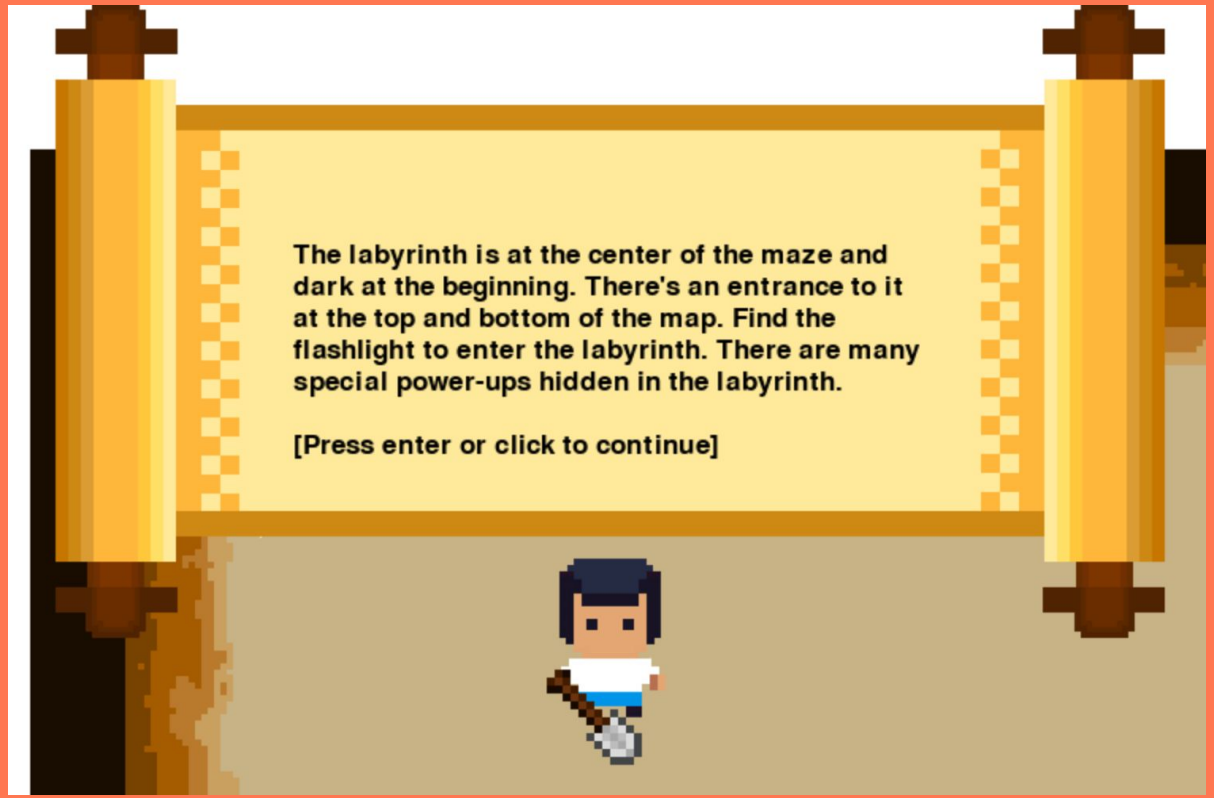

Use the arrow keys to move around the labyrinth.
Pick up weapons by pressing "d" and drop items
by pressing "f". You can use these items
by pressing the "space" key.
[Press enter or click to continue]

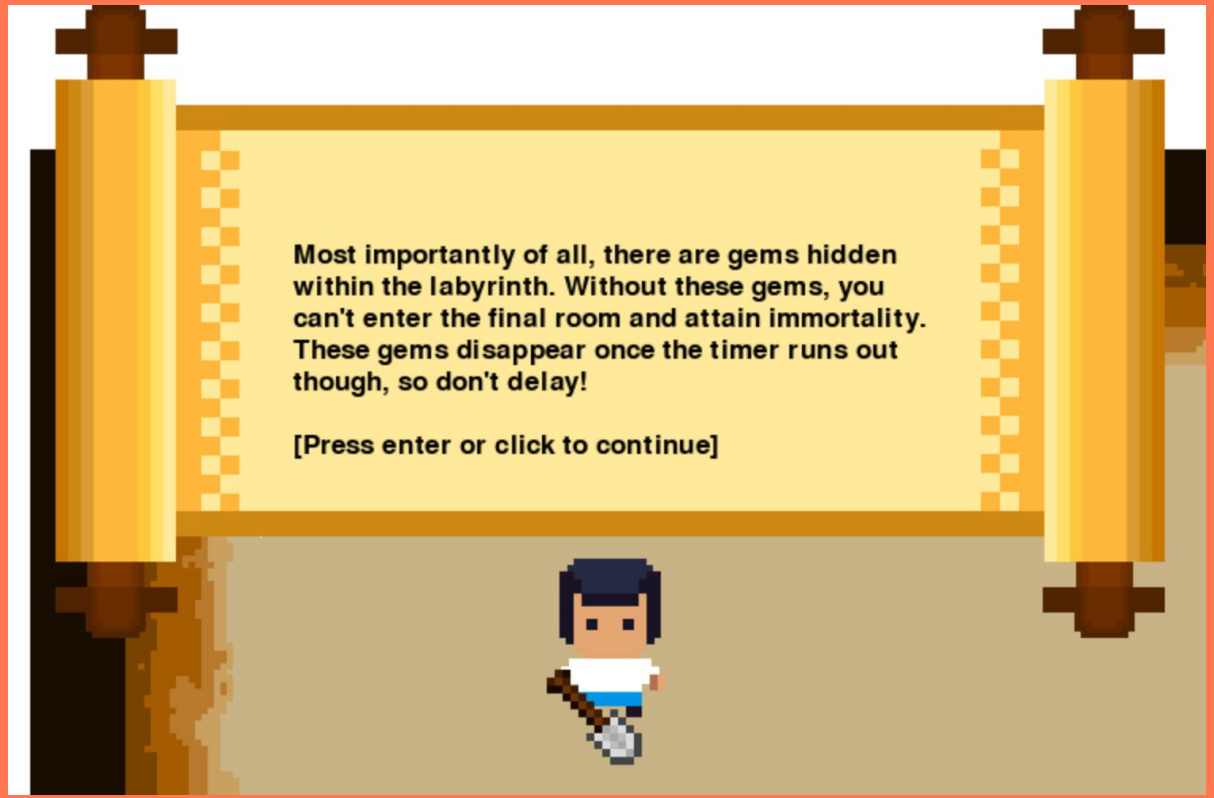

Kill the monsters using weapons. Each of the six monsters have their own room, and they automatically leave their room once the previous monster has been killed.
[Press enter or click to continue]


Beware: the monsters have many special powers,
one of which is tracking you down. Unlike you,
they are also able to walk through objects (although
they have not yet evolved to walk through walls).
[Press enter or click to continue]


If you are not using a weapon, you can place it
into your backpack by pressing "e" and pressing "r"
to cycle through the items in your backpack.
You can keep a maximum of five weapons in
your backpack at any one point.
[Press enter or click to continue]


You can view the menu of items you have by pressing "a". From this menu, you can view the stats of all the weapons as well as selecting which weapon you wish to equip by pressing a number 1-5. Close the menu by pressing "s".
[Press enter or click to continue]


The labyrinth is at the center of the maze and dark at the beginning. There's an entrance to it at the top and bottom of the map. Find the flashlight to enter the labyrinth. There are many special power-ups hidden in the labyrinth.
[Press enter or click to continue]


Most importantly of all, there are gems hidden within the labyrinth. Without these gems, you can't enter the final room and attain immortality. These gems disappear once the timer runs out though, so don't delay!
[Press enter or click to continue]

The Basics

Arrow Keys: Move

“F” Key: Drop Object

“E” Key: Place Object in Backpack (not in hand)

Esc Key: Quit the Game

Space: Use Object (see later slides)

“D” Key: Pick Up Object

“A” Key: Open Menu of Items

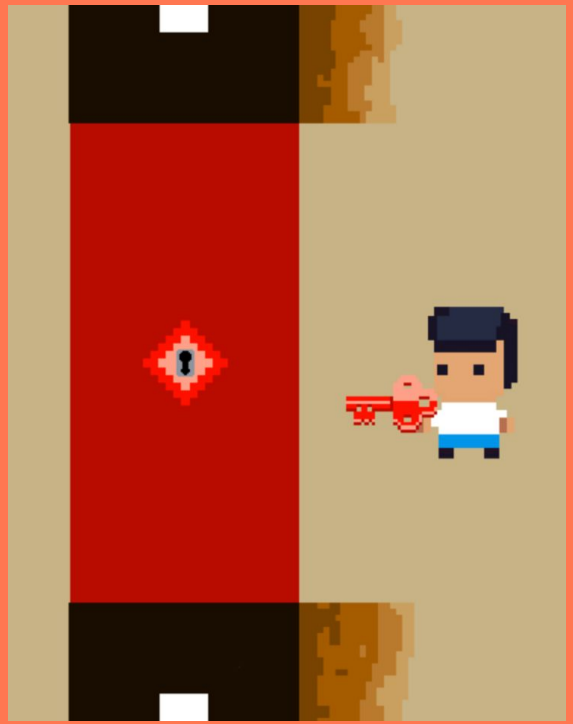
Eating Plants (for Health Points)

Use the shovel to dig up the plant in order to gain 20 health points. Remember, you can only have a maximum health of 100, so don't try to dig up a plant when you are at maximum health.

[Press enter or click to continue]

Space *Use space key to eat the plant*

Opening Doors

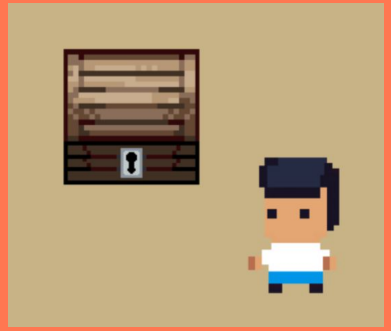


Congratulations, you just killed your first monster!
You can use the key to access the monster's rooms in order to find treasure and stronger weapons.
The key can only open the correct door, and the keys and doors are color coded.

[Press enter or click to continue]

Space *Use space key to open the door*

Using Chests



Press "w" to open or close the chest. Each chest contains a treasure of some kind, usually a powerful weapon that can help you kill monsters. Remember, you can view weapon stats by pressing "a" and looking at the menu of items.

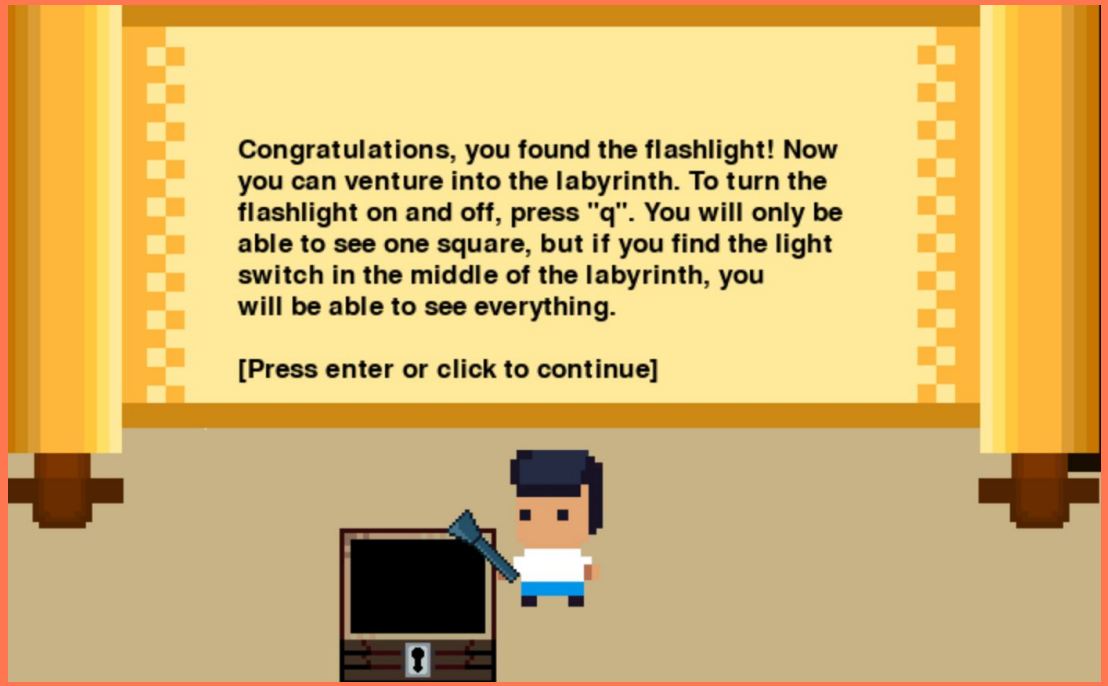
[Press enter or click to continue]

To toggle (open/close) chest

To pick up the weapon/object

Flashlights

To toggle (turn on/off) flashlight



Light Switch

Click the light switch to permanently turn on the light in the labyrinth. Now, you won't need to hold the flashlight in order to see.

[Press enter or click to continue]

To toggle button

Gems/Pieces

Congratulations, you found your first gem.
Collect at least five in order to access the final room and attain immortality. Keep an eye on the clock though, and make sure that time doesn't run out first!

[Press enter or click to continue]

End of the Game

- Once the player picks up the **potion/elixir of immortality** in the final room, this animation plays
- The player then wins the game and is given the option to close the game window

Map/Floor Plans

The Impenetrable Labyrinth

Map & General Features

Note: not to scale

Map is 8000x8000 pixels; **rooms** are 405x630 pixels

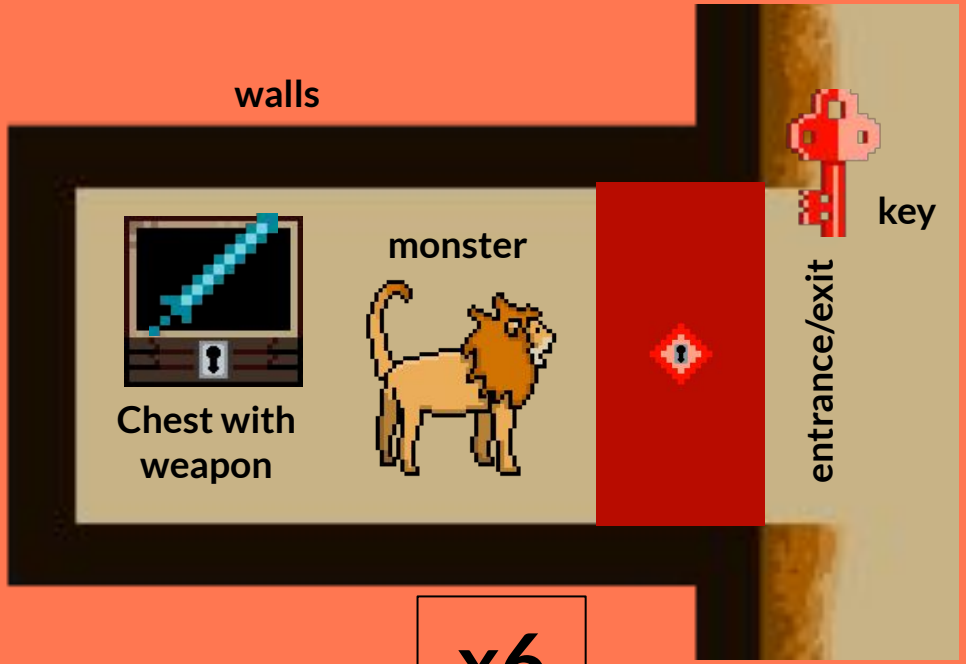
Shown

- **Chests** - holds weapons/objects
- **Plants** - for gaining health points
- **Gems/Pieces** - collected to open last door (locations randomly assigned for each new game) (disappear after 8 minutes)
- **Doors** - leads to the chests and monsters
- **Light Switch** - lights up the labyrinth
- **Potion** - immortality/end of the game
- **Player** - the user's sprite (starts off holding shovel)
- **Monster** - attacks the user and reduce their health points
- **Weapons** - utilized by player to attack the monsters (a new weapon appears in the monster's chest after the monster is killed)
- **Keys** - for unlocking doors (appear after the corresponding monster is killed)

Not Shown

- **Darkened Labyrinth**
- **Menus, Instructions, Text, etc.** - displays information

Physical Rooms

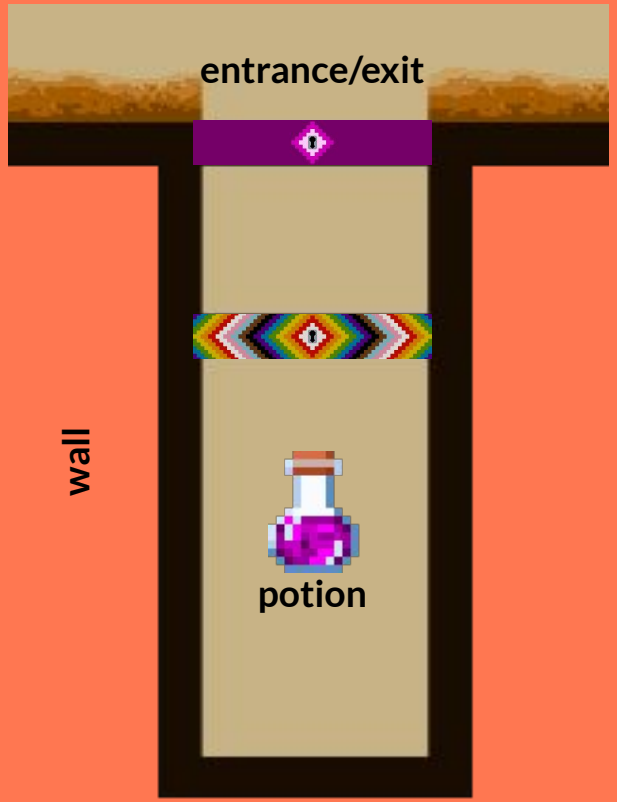


x6

Each of the 6 Rooms on the sides

- 405 pixels by 630 pixels
- Contains a **chest** that holds a special **weapon**
- Is locked by a **door (entrance/exit)** that can only be opened by the corresponding **key** (color coded)
- Releases a new **monster** when an old monster is defeated
- **Walls** cannot be penetrated

Physical Rooms



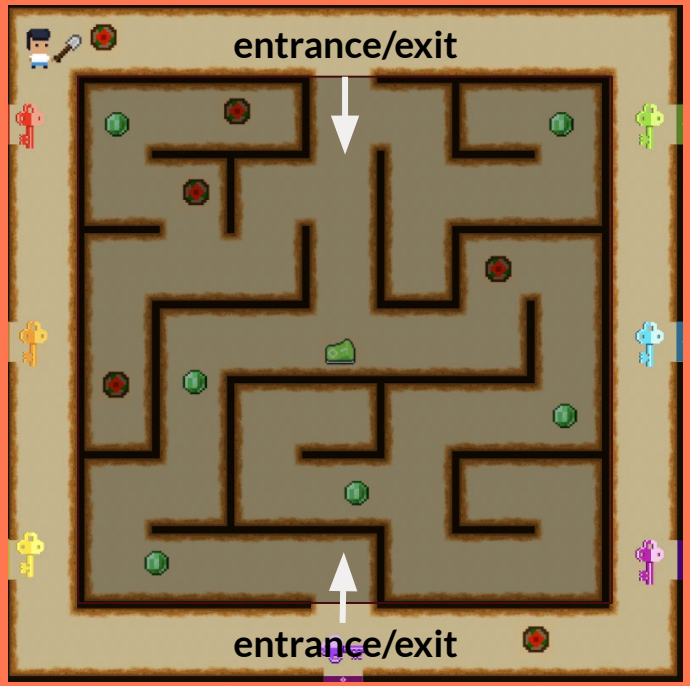
The Final Room

- 630 pixels by 405 pixels with 265 pixel corridor extension
- Contains the **potion/elixir of immortality**, which the player has to drink to win the game
- Is locked by a normal **door** (pink, opened by pink key) but also a **special rainbow door** that can only be opened if the user has collected at least 5 **gems**
- **Walls** cannot be penetrated

Opening the Last Door

- Rainbow door opens automatically when at least 5 gems are placed in the area in front of it

Physical Rooms



The Labyrinth

- 2 **entrances/exits**
- Starts off as completely dark 
- Can be illuminated by the **flashlight** or permanently lighted by the **light switch** (but the player has to find the light switch first) 
- Contains **plants** for rejuvenation 
- And **gems** for collection 
- Walls cannot be penetrated

Rooms/Classes

*Note: for our game, our rooms take the form of classes representing different parts of the game, and they are applicable throughout the game. The Detailed Floor Plans were done for the physical rooms. This section contains descriptions and functions for each ‘class’.

Background

- The backdrop/screen of the game (the labyrinth)
- Used when the player interacts with and makes changes to the background (e.g., dropping weapons, lighting up rooms, opening doors)
- Created by: Spencer

Variables

Self.surface: size of surface/map to draw on is 8000x8000

Self.sizex: main pygame screen width

Self.sizey: main pygame screen height

Self.screen: main pygame screen

Self.wall_list: set of wall points

Self.wall_cells: dictionary of which cells have which walls (north, east, south, and/or west) in the labyrinth grid

Self.door_list: list of all door objects in the background

Self.background_obj: list of all objects in the background

Variables

Self.stagePosX & self.stagePosY: Starting center point position (the player's position at the beginning of the game)

Self.offset_x and Self.offset_y: Offset used to when drawing items into background to ensure proper placement

Self.image: background image

Self.width & self.height: width and height of background image

Self.dark: image for the dark sections of the labyrinth

Self.light_switch: light switch button that can be clicked on to turn on all lights in the labyrinth (a BackgroundButton object)

Background

__init__(self, sizex, sizey, bg_img, stagePos, offset=(505, 390), light_switch=(0,0))

- self: the background itself
- sizex: horizontal dimension of the screen on which the game is displayed
- sizey: vertical dimension of the screen on which the game is displayed
- bg_img: filename of image to be used for the background
- stagePos: tuple of the x and y coordinates the player starts at
- offset: tuple of x and y offsets when drawing items like walls; used to ensure proper placement in background; set to (505, 390) by default
- light_switch: tuple of light switch x and y coordinates in labyrinth; set to (0,0) by default

Initiates the background of the game (namely, the pre-drawn labyrinth image)

Background

def one_wall(self, x_left, y_top, x_right, y_bottom)

- Places a single rectangular wall onto the screen (taking in the coordinates of the rectangle)
- Called by place_walls(self) for each specific segment of the entire wall system

def place_walls(self, h=405, w=630, hor_rooms=None, vert_rooms=None, corridor_room=None)

- Contains a dictionary indicating where all the walls are and the dictionary is fed into one_wall(self, x_left, y_top, x_right, y_bottom) to initiate a wall
- The entire labyrinth is divided into 9 by 9 grid and the walls are defined relative to each of the box in the grid
- Walls for individual rooms are also made based on optional parameters, including room height, width, and locations (from the lists hor_rooms and vert_rooms)
- The walls are extended in front of corridor_room

Background

def set_background_image(self):

- Redraws the background when there is a change made to it, such as removing or dropping a weapon. In these cases, the background needs to be redrawn.
- The process: backdrop, doors, objects that are not monsters, offset

def scroll(self, x, y, player, weapon=None)

- x, y → which way is scrolling - for example, x = 5 scrolls horizontally to right
- Checks if the player is allowed to move (e.g., if the player is in contact with anything)
- Moves by 5 pixels (redraw) if player is allowed to move
- Does not move if player is not allowed to move

def detect_wall_collision(self, player)

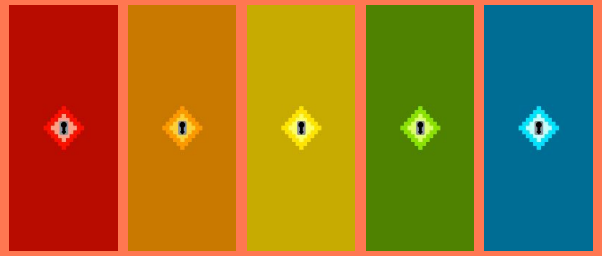
- detects if the player's x and y points are in a wall by looking for an intersection in these coordinates.
- Returns True if colliding, False if not

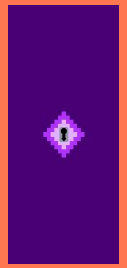
Background

def monster_detect_wall_collision(self, monster) • detects if the monster's x and y points are in a wall by looking for an intersection in these coordinates. • Returns True if colliding, False if not
def item_detect_wall_collision(self, item) • detects if a weapon's x and y points are in a wall by looking for an intersection in these coordinates. • Returns True if colliding, False if not
def def make_doors(self, main_x=(1093,6584), main_y_top=1824, door_dist=1740, corridor_doors=[(3756, 6392), (3756, 6664)]) • Creates doors via the Door class and adding them to a list • 3 doors are made on each side of the labyrinth along with 2 doors in the last room's corridor

Background

def place_doors(self) • Placing the doors (calling from the list) with the place() function
def place_dark(self, player, all=False) • Contains all places defined to be dark and calls place_one_dark(self, row, col) to place them
def place_one_dark(self, row, col) • Creating the dark spots in the middle of the labyrinth that is meant to be illuminated by flashlight • row & col corresponds to which row and col in the 9 by 9 grid to turn dark
def get_new_loc(self, object) • retrieves an object's grid row and grid column coordinates in the 9x9 labyrinth grid • for plants/chests/etc when displaying story/instructions





Door

- Controls all the doors throughout the game
- Changed and called when the player interacts with the door (e.g., opening, colliding, etc.)
- Created by: Spencer

Variables

Self.images: all of the open and closed door images in a list

Self.width & self.height: door width and height

Self.screen: Background object that the door is a part of

Self.frame: frame number of door; 0 is closed; 1 is open

Self.x_bg & self.y_bg: x & y positions of door with respect to the background

Self.x & self.y: x & y positions of the door with respect to the pygame window

Self.passable: set of the points of the door that should be removed from the labyrinth walls if opened and the points that should be added if closed; starts off empty and is populated as needed

Door

def __init__(self, screen, x, y, pos, frame=0) • Initiates a door with specified location with respect to the background and indicated frame (indicates whether the door is open or closed based on the sprite in use)
def place(self) • Places the door onto the screen
def get_passable(self, open=True) • Unlocks (makes passable) or locks (makes unpassable) the door region based on the value of open, which is passed in by other functions like openDoor(self) and closeDoor(self)

Door

def open_door(self) • opens door • calls get_passable(self, open=True) with open=True (default value)
def close_door(self) • closes door • calls get_passable(self, open=False) with open=True (default value)
def open_with_items(self, items_list, min_num, background, corridor_length=310) • opening the last door with the 5 pieces • ensures that the pieces are places within a certain distance (specified as corridor_length) from the door • makes sure that the player has all 5 pieces and then opens the door

Button(pygame.rect.Rect) & BackgroundButton(Button)

- A class(es) for creating buttons - such as the pause/continue and light switch buttons
- Created by: Joy

Variables

Self.images: all of the button sprite images in a list

Self.width & self.height: button width and height

Self.frame: button frame

Self.x & self.y: x & y coordinates of the button with respect to the pygame window

Self.screen_loc: screen to draw button on

Self.nick: nickname

Self.count: counter that implements a lag to ensure that the button click only registers once

Self.prev_clicked: whether the button has already been clicked

Button(pygame.rect.Rect)

def __init__(self, x, y, screen_loc, nick) ● Initiates a button in the Button class with size, location, and nickname
def place(self) ● Placing the button onto the screen
def detectClick(self, background=None) ● Returns True if a mouse click on the button is detected (checks if mouse click is within the ranges of the x and y coordinates of the button)

BackgroundButton(Button)

def __init__(self, x, y, screen_loc, nick)

- Initiates a button, calling the Button super class with size, location, and nickname
- Initiates self.walk_over = False
  - a Boolean variable that dictates whether or not the player can walk over something that's a background object.
  - walk_over=True for all the weapons
  - walk_over=False for plants and chests → stops the player when the player collides with them

def detectClick(self, background)

- Returns True if a mouse click on the button is detected (checks if mouse click is within the ranges of the x and y coordinates of the button)

def detect_collision(self, other)

- Detects a collision between the weapon (self) and another object/sprite (other) by checking the coordinates of each parameter

Chest(pygame.surface.Surface)

- Controls all the chests throughout the game
- Changed and called when the player interacts with the chests (e.g., opening, taking weapon out, etc.)
- Created by: Amanda

Variables

Self.x_bg & self.y_bg: x & y coordinates of the chest

Self.images: list of possible image frames for chest (e.g., open, close)

Self.width & self.height: width and height of the chest

Self.content: what is contained within the chest

Self.walk_over: whether the chest can be walked over (False)

Self.frameL current frame displayed (closed)

Self.state: true if open, false if closed

Self.lag: lag count for toggling

Chest(pygame.surface.Surface)

def __init__(self,x_bg,y_bg) ● Initiates a Chest object from the Chest class with passed in location coordinates ● Calls and initiates with the super class pygame.surface.Surface
def place_object(self,object) ● Places the passed-in object into the chest; called in game.py to let player store items
def pick_up_object(self,player,background) ● Opposite of place_object(self,object)--let the player take the object in the chest out ● Calls the Weapon class function pick_up(self, background, player) to pick up

Chest(pygame.surface.Surface)

def toggle(self,player,background) • Opens and closes the chest • Function is called via keyboard input • Calls the open(self,player,screen) and close(self,player,screen) functions based on the value of self.state (open if False, close if True)
def open(self,player,background) • Opens the chest if player is touching the chest (calling touching(self,other,screen,tolerance=0,player=True)) and changes self.state = True and self.frame = 1 to enable toggling
def close(self,player,background) • Closes the chest if player is touching the chest (calling touching(self,other,screen,tolerance=0,player=True)) and changes self.state = False and self.frame = 0 to enable toggling

Chest(pygame.surface.Surface)

def place(self,background)

- Placing the chest (self) onto the screen

def touching(self,other,background,tolerance=0,player=True)

- Checks if the chest is being touched by another object (other) by comparing the coordinates of the two objects.
- *A tolerance level (applicable to other functions as well) is set up to alter the coordinate systems slightly so the objects have to be closer or further to be considered as touching. This addition would eliminate the possibility of sprites getting stuck together.
- Returns True if ‘touching,’ False if not

GAME

- Sets up the entire game
  - Lets the user move and interact with the objects via keyboard and mouse inputs
  - Sets up the menus, instructions, levels, locations, displays, etc.
- Created by: Spencer, Amanda, Joy (this was the most major part of the code, so we all worked on it based on the separate classes we wrote)

Variables

Background

Self.screen: main screen of the game

Self.frame: current frame number

Self.buttons: list of all buttons

Self.labyrinth_lights_on: whether or not the labyrinth should be fully visible

Self.pause: whether the game is paused or not

Characters

Self.player: player object

Self.monsters: list of all monsters

Variables

Self.active_monster: monster for current level, initially set to None

Items

Self.weapons: list of all items the player can hold (except keys)

Self.available_weapons: list of available items the player can hold (i.e., all items the player has 'found'), depends on level

Self.keys: list of all keys

Self.potion: potion

Self.items_list: list of all items the player is holding (backpack)

Self.extend: whether the item the player is holding should be extended

Variables

Background Interactable Objects

Self.plants: list of all plants

Self.chests: list of all chests

Self.rooms: horizontal and vertical rooms to be placed in the game

Story & Text

Self.font: font of text

Self.text: text that should be displayed on screen

Self.story: storyline objects

Variables

Self.instruction_text_num: tracks what slide of the instructions screen is present

Self.introductions: a dictionary (string: boolean) that tracks whether the player have encountered a series of objects for the first time yet, and if not, pops up with instructions

Self.final_story_instructions: when final story instructions should be displayed or not

Levels

Self.level: current level number

Self.max_levels: total number of levels, determined based on the number of monsters

Self.tutorial: whether the game is in the tutorial or not

Self.complete: whether the game is complete; whether the last level has been beaten

Variables

Menu

Self.menu: the menu for displaying items in the backpack

Self.menu_pause: whether the game was paused in order to display the menu (displaying the menu pauses the game for that, but just clicking the pause button doesn't display the menu)

Counters

Self.flashlight_lag: lag for turning flashlight on/off

Self.damage_lag: lag for doing damage to monster

Self.items_list_lag: lag for looping through items list

Self.c: animation frame counter

Variables

Clock

Self.clock: clock for game

Gems

Self.gems: list of all gems

Self.collected_gems: tracks how many gems the player has collected so far

Self.time_left: time left until gems disappear; 8 minutes

Self.fps: frame rate of game; 120 fps

Self.countdown: countdown object to be drawn on the screen

Variables

Flashlight

Self.flashlight: for using the flashlight

Error Messages

Self.error: whether error message should be displayed

Self.error_msg: the content of the error message

Selfbp_error_type: type of error if occurring with backpack items

Pygame Events

Self.events: pygame events (e.g., key clicks)

Game

def __init__(self,screen,player,weapons,monsters,keys,pieces,plants,buttons,text,login,story, chests,items_list,level=0,frame=0,labyrinth_open=False,count=0,k_count=0,z_count=0, q_count=0, tutorial=True,pause=False,extend=False) • Starting the game with all the monsters, weapons, buttons, plants, etc., which should be previously initiated prior to being passed in • Contains lag counts for keyboard input and other minor variables for controlling the game
def setup(self) • Sets up the first level of the game—drawing the player, placing the walls, pieces/gems, weapons, plants, monsters, chests, health bar, etc.

Game

def gem_countdown(self) • Sets up and displays the timer for the game; removes gems from labyrinth once the timer runs out
def random_gem_locations(self) • Randomly assign locations to “hide” the gem pieces that are to be collected by the player • Called in setup(self) to set up the gems
def make_menu(self) • Creates the menu without actually drawing it, done so that the menu isn't remade during every frame • Creates objects from the WeaponSlide, Stats, and Menu classes
def menu_display(self) • Draws the menu on screen • Allows the menu to be hovered over (color change), clicked (display more information), closed (pressing ‘s’ key), etc.

Game

def story_screen(self) • calls story_display(self,text="") to display text
def story_display(self,text="") • displays the story text • using the strings of text in the text.py file by default
def instruction_screen(self) • displays instructions in the screen
def introduction_screen(self) • calls functions to display instructions on screen for first encounters of objects like plants, keys, etc
def check_introduce (self, obj, str) • checks if an item is being encountered for the first time and introduces it if yes
def introduce (self,str) • displays instructional text for an item on its first encounter

Game

def manipulate_backpack(self)

- if the player is holding an item and presses 'f', the item is dropped
- if the player is not holding any items and presses 'd', the item that they are on top of (if any) is picked up
- if a weapon is in hand and ‘e’ is pressed, the item is placed in backpack
- if the backpack is not empty and ‘r’ is pressed, the program cycles through backpack items (holding a different weapon every time)

def item_interaction(self)

- if the player is holding the flashlight and pressed 'q', the flashlight is turned off if it's on and on if it's off
- if the 'q' key is held, the state is changed every 80 frames
- if ‘w’ key is pressed, the chest is toggled open and close
- if ‘d’ key is pressed, the player is not holding a weapon, the chest is not empty, etc., the object in the chest is picked up
- digs up a plant if a shovel is being extended and is in contact with the plant
- opens a door if the key that matches the door is extended and is touching the door
- turns off the flashlight if the player is not holding it

Game

def place_player(self) • draws the player behind the weapon if the player is not facing up • draw weapon on player if holding weapon • draw the player in front of weapon if the player is facing up • retrieves the new grid row and column position of the player
def move_player(self) • moves the player by shifting the background in the other direction • arrow keys on keyboard dictate the direction and the direction the sprite is facing
def darken_labyrinth(self) • if the labyrinth is not fully lit, the flashlight is on, and the player is inside, lights up only the grid square the player is in • if the labyrinth is not fully lit and either the player is not in it or the flashlight is not turned on, the labyrinth is fully dark

Game

def place_monster(self)
- if there is an active monster, have the monster track the player
- retrieves the new grid row and column position of the monster

def monster_interaction(self)
- if the player is using an item (if the item is extended) and that weapon comes into contact with the monster, does damage to the monster
- alerts the used that they cannot do damage to the monster with the key or the flashlight
- does damage to the monster either each time the weapon is extended or every 100 frames if the weapon is left extended
- if the monster dies, proceeds to next level

def update_health_bars(self)
- updates health bars for monster and player with new number of health points

Game

def pause_screen(self) ● pauses the game (no more interaction allowed except for clicking play button to resume)
def tick(self, fps, events) ● uses frame rate to increment game clock
def text_update(self) ● set up instruction text in the bottom right corner informing player of controls ○ always displays basic instructions like dropping/picking up weapon, closing game, pressing space to extend, etc ○ when the player wields a flashlight, display instructions for turning flashlight on and off
def next_level(self) ● proceeds the game to the next level by setting up/updating the corresponding key, level (calling levels(self) and/or final_level(self) (if on last level)), background, monsters, weapons, health bars, etc.

Game

def levels(self) • change monster and weapons based on level
def final_level(self) • sets up the final level, in the same way as self.levels() but specifically catered to the last level
def win_end_screen(self) • displays storyline for winning game
def lose_end_screen(self) • displays storyline for losing game
def beat_game(self) • displays animation for winning game

HealthBar(pygame.surface.Surface)

- Displays the health points of the player and current monster throughout the game
- Changed and called when the player and/or monster suffers health damage
- Created by: Amanda

Variables

Self.value: current number of health points

Self.max_value: maximum number of health points

Self.x: x coordinate of bar

Self.y: y coordinate of bar

Self.nick: nickname of character the bar represents

Self.text: text box located in bar

Self.text.text : text to place on top of bar; shows number of points out of the maximum

HealthBar(pygame.surface.Surface)

def __init__(self, value, x, y, nick) ● Initiates a health bar with its value, location, and nickname ● Utilizes the super class pygame.surface.Surface and defines additional items
def place(self,screen) ● Places the text and the health bar onto the screen
def update_bar(self,screen, new_val=None) ● Updates the bar based on current status of the game ● Calls the place(self,screen) to renew the display of the text and the bar on the screen

Menu (pygame.surface.Surface)

- When created, displays all items that the player is in possession of and lists them
- Allows the user to select an item by clicking the corresponding number
- Allows the user to view weapon stats (how many damage points each weapon does to each monster)
- Created by: Spencer

Variables

Self.x & self.y: x and y coordinates of menu

Self.width & self.height: width and height of menu

Self.slides & Self.sub_windows: all item slides and stats windows

Self.padding: padding between slides

Self.title_text: title of menu

Self.selected: the selected item; the stats window is displayed for this item

Self.redraw: whether the menu needs to be redrawn or not

Menu (pygame.surface.Surface)

def __init__(self, x, y, title, menu_items, width, height, padding=20) • Initiates a menu with the given parameters, such as title, the items in the menu, dimensions, padding
def build(self) • creates the menu with the title and items with pygame.surface.Surface, TextBox, and WeaponSlide objects
def draw_all_items(self) • Creates and draws all WeaponSlide items in the item container with the specified padding between slides
def place(self, screen) • draws the menu but does not modify its contents

WeaponSlide(pygame.surface.Surface)

- Creates each row/object displayed on the menu
- Created by: Spencer

Variables

Self.nick: slide nickname

Self.image: image to be shown on slide

Self.x & Self.y: x and y coordinates of slide; both 0 by default and are updated as the slide is drawn

Self.width & self.height: slide width and height

Self.padding: padding between items in slide

Self.val: number of the slide in the menu

Self.bg_color: background color of slide

Self.stats_btn: triangle showing whether stats are being displayed for the item or not

Self.displaying_stats: whether stats are being displayed for the slide or not

WeaponSlide(pygame.surface.Surface)

def __init__(self, nick, image, width, height, val, bg_color=(225, 193, 110)): ● Initiates a WeaponSlide object with given parameters
def make(self) ● Sets up the display for the slide, with appropriate images, background, text, etc. ● Called any time an update to the slide needs to be made (e.g., making color lighter on hover)
def hover(self) ● Checks if the user is hovering over the slide ● Returns True if hovered
def detectClick(self) ● checks if the user has clicked the slide ● Returns True if clicked

Stats(pygame.surface.Surface)

- Displays (as an extension of the menu/backpack) the damage points of the selected weapons
- Accessed when the corresponding WeaponSlide is clicked on
- Created by: Spencer

Variables

Self.nick: stats window nickname

Self.width & self.height: stats window width and height

Self.x & Self.y: x and y coordinates of window

Self.padding: padding around text in window

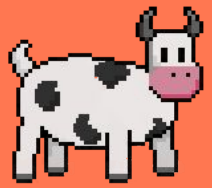
Self.stats: dictionary containing stats

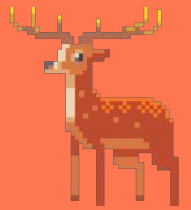
Stats(pygame.surface.Surface)

def __init__(self, nick, width=300, height=400, x=600, y=300) ● Initializes a Stats object for display
def get_stats(self) ● Retrieves game stats of the item corresponding to the stats screen
def place(self, screen) ● Calls get_stats(self) and place stat information in the form of text / TextBox object

Monster(Player)

- Controls all 6 monsters throughout the game, including
  - Moving
  - Attacking & Tracking Player
  - ‘Twitching’ when attacking the player
- Created by: Amanda





Variables

Self.nick: nickname

Self.state: whether or not the monster is active (True is active, False is inactive)

Self.x_bg, & self.y_bg: x & y positions in relation to the entire background

Self.attack_pts: how much an attack is worth

Self.speed: number of pixels the monster moves in one move

Self.frame: the direction the monster is facing (different sprite for each direction)

Self.walk_over: whether the player can walk over it

Self.move_rate: how often it moves, probability between 0 and 1, inclusive

Self.previous: the previous move (string "left","right","up","down"); used so there is a bias for moving in the same direction

Self.collide: direction that the monster is colliding with the player in; string: ("left", "right","up","down") or None

Self.n: attack delay counter

List of Monster Subclasses

01

Lion

attack_pts = 5

02

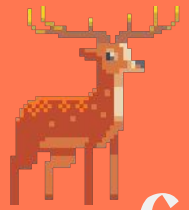
Cerberus

attack_pts = 6

03

Hydra

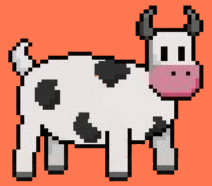
attack_pts = 7



04

Golden_Deer

attack_pts = 8



05

Cattle

attack_pts = 9

06

Boar

attack_pts = 10

Monster(Player)

def __init__(self, nick,x_bg, y_bg, attack_pts, speed=5, state=False)

Initiates a Monster object with

- nick: nickname
- x_bg: x-coordinate in relation to the entire background
- y_bg: y-coordinate in relation to the entire background
- attack_pts: power (in points) of a single attack
- speed: speed of the monster, defined as the number of pixels the monster moves in one move
- state: whether or not the monster is active (True is active, False is inactive)

Other variables:

- self.frame = 1
- walk_over = False
- self.move_rate = 0.75: how often it moves, probability between 0 and 1, inclusive
- self.previous = None: the previous move (string "left","right","up","down") → used so there is a bias for moving in the same direction
- self.collide = None: direction that the monster is colliding with the player in; can be "left", "right", "up", "down", or None
- Initiates via the Player superclass

Monster(Player)

def track_player(self, background, player, game)

- Makes the monster move in the direction of the player
- The closer the monster is to the player, the more likely it is to run towards the player → based on vectors of probability and the vector between the monster and the player
- The monster cannot travel through walls but can move through other background items for increased challenge
- Calls the move_left(self,background,direction), move_right(self,background,direction), etc. functions to execute motion

def move_left(self,background,direction)

- Move the monster to the left and updates the monster image
- Unless running into a wall, which would result in a up or down motion depending on the situation
- Direction: list of probabilities for each direction, in the order: left, right, up, down

Monster(Player)

def move_right(self,background,direction)

- Move the monster to the right and updates the monster image
- Unless running into a wall, which would result in a up or down motion depending on the situation
- Direction: list of probabilities for each direction, in the order: left, right, up, down

def move_up(self,background,direction)

- Move the monster up and updates the monster image
- Unless running into a wall, which would result in a left or right motion depending on the situation
- Direction: list of probabilities for each direction, in the order: left, right, up, down

Monster(Player)

def move_down(self,background,direction)

- Move the monster down and updates the monster image
- Unless running into a wall, which would result in a left or right motion depending on the situation
- Direction: list of probabilities for each direction, in the order: left, right, up, down

def place(self,background,extend=False)

- Placing the monster on to the background and extending if the extend value passed in = True (by calling the extend(self) function)

def extend(self)

- “Twitching” motion of the monster when it attacks the player
- Shifts the monster over by 10px, direction depending on what direction the monster is facing.

Monster(Player)

def touching(self,other,background,tolerance = 0)

- Checks (with passed-in tolerance) if the monster is being touched by another object (other) by comparing the coordinates of the two objects.
- Contains several cases where ‘other’ could be a door, the player, wall, etc.
- Returns True if ‘touching,’ False if not

def attack(self,background,player,game)

- Does damage to the player if the player is not attacking the monster
- The monster has to be ‘touching’ the player in order to reduce the player’s (passed-in ‘player’) health; calls the decrease_health() function on the player object

Plant

- Plants can be dug up/eaten by the player to gain health points
- Created by: Amanda

Variables

Self.x_bg & self.y_bg: plant x & y coordinates

Self.frame: current frame of plant (changes as it is dug by the player)

Self.walk_over: False - plant cannot be walked over by the player

Self.dig_lag = 0: digging lag so that, if digging continuously, plant is only dug after a certain number of frames

Self.images: list of sprites for the plant

Self.width & self.height: width and height of plant

Plant

def __init__(self, x_bg, y_bg) ● Initiates a Plant object with passed-in locations (x_bg, y_bg) ● Save the images of the plant into the images list
def draw(self, background) ● Draws the plant onto the background
def eat(self, player, background) ● Lets the player ‘eat’ the plant to gain health by 20 points ● The plant would be removed from the background once eaten and the background renews itself

Player

- Controls the player throughout the game, including
  - Moving
  - Attacking monster
  - Using weapons
  - Gaining or losing health points
  - Becoming immortal at the end
- Created by: Spencer

Variables

Self.images: list of images for each frame of the character; there is 1 frame for each direction the character faces (up, down, right, or left)

Self.width & self.height: width and height of the character

Self.x & Self.y: x and y positions in reference to the window; main player will always be in the middle of the screen

Self.held_item: item that the character is currently holding in their hands

Self.possible_weapons: list of items that the character has acquired

Self.items_list: list of items the character has in their backpack or in their hand

Self.health: health points; starts at 100; this is also the maximum value

Self.pos_row & Self.pos_col: grid cell row and grid cell column numbers of character in the 9x9 labyrinth grid

Variables

Self.hit: whether or not the character is being attacked; initially, the character is not being hit

Self.hit_count: implements a lag for being attacked (so that points are subtracted less frequently than each frame)

Self.animation_images: dictionary images for character animation

Self.image: image of character animation if the animation is running

**animation_images and image are only used if the character is the main player and has beat the game

Player

def __init__(self, filename, items_list=None)

- Initiates the player
- The function retrieves the images for each frame of the character. There is 1 frame for each direction the character faces (up, down, right, or left), and the name of the image file corresponds to ‘filename.png,’ where ‘filename’ is passed into the function and n is an integer.
- Items_list keeps track of what's in the backpack & what's in the hand; the player starts off with an empty list when player is first initiated

def get_new_loc(self, background)

- Retrieves the player’s new location (i.e., grid row and grid column) in the 9x9 labyrinth grid

def place(self, background, frame)

- places the player on screen and draws the player facing in the specified direction, indicated by frame
- Also makes sure that the player turns red for a short time when its health is getting damaged

Player

def touching(self, other, background, monster = True)

- Checks whether the player is touching a particular object (other)
- If yes, returns True; if not, returns False
- There are several cases:
  - Monster = True: other is a monster
  - Background object
  - Not a monster and not a background object

def decrease_health(self, pts)

- Decreases the player's health points by the specified amount (pts)
- Sets self.hit to True to show the player has been hit

def increase_health(self, pts)

- Increases the player's health by the specified number of points, but only if this value will not exceed 100

Player

def usingItem(self, weapon, extend)

- Checks whether or not the player is holding a particular weapon and whether or not that player has been extended
- If both conditions are met, returns True; otherwise, returns False

def become_immortal(self, frames=32, delay=25)

- Obtains animation images for the program to run through to show the player becoming immortal and ending the game

Textbox(pygame.surface.Surface)

- Displays text messages in a box on screen
- Created by: Amanda

Variables

self.x & self.y = x and y positoons

Self.width & self.height = width and height of the textbox

Self.text = the text to be displayed itself

Self.text_size: size of the text

Self.font: font of the text

Self.rect: rectangular background of the text

Self.padding : padding for the text

Self.top_padding: top padding for the text

Self.left_padding: left padding for the text

TextBox(pygame.surface.Surface)

def __init__(self,x,y, width, height, text_size=25, text="", padding=None, top_padding=0) • Initiates a textbox with given location (x,y), dimensions (width, height), text size, etc.
def place(self, screen,background_color=None,text_color=(0,255,0)) • Places the text box onto the screen
def updateText(self, background, new_text=None) • Updates the text with new_text and renews the background.
def detectClick(self) • Returns True if a mouse click on the textbook is detected (checks if mouse click is within the ranges of the x and y coordinates of the button)

ImageBox(TextBox)

- General class for displaying text-containing images that convey information to the player
- Created by: Amanda

Variables

Self.image: image to be displayed

Self.width & Self.height: dimensions

ImageBox(TextBox)

def __init__(self,x,y)

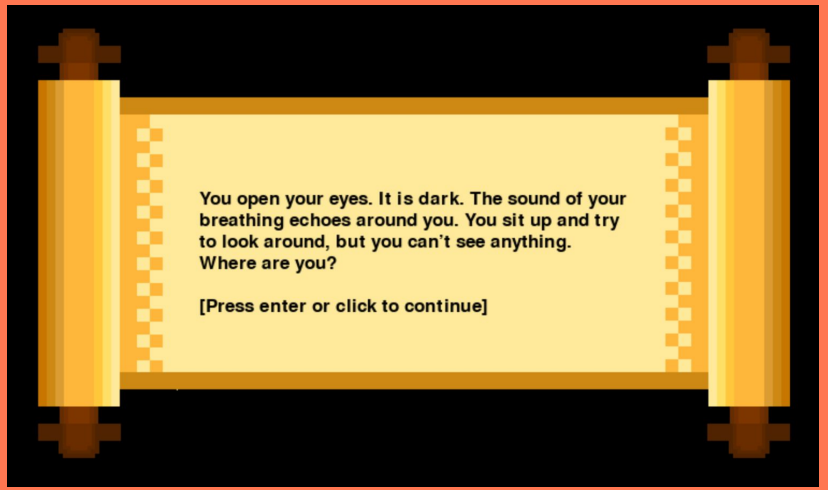
- Initiates an ImageBox object (subclass of Textbox) with given location (x,y)
- (to make pop-ups for various purposes, such as informing players of the storyline)

def place(self, screen,text_color=(0,255,0))

- Places the image box onto the screen

Story(ImageBox)

Displays storyline messages
Created by: Amanda



Variables

Self.story_text: text explaining the storyline of the game

Self.instruction_text: text explaining the instructions on how to play the game

Self.introduction_text: Text introducing different kinds of objects when the user finds/encounters them

Self.final_level: text informing the player that they reached the end of the game

Self.win_end_screen: text informing winning

Self.lose__end_screen: text informing losing

Story(ImageBox)

def __init__(self, x, y) ● Initiates the storyline pop-up texts
def story_display(self, screen, text, choices = None,next = "[Press enter or click to continue]") ● Display the text on the screen ● Choices will be a dictionary of what keys can be pressed and what the next screen will be

ErrorBox(TextBox)

Displays error messages at the bottom of the screen
Created by: Joy

Variables

Self.count: frame counter to keep the error box visible for only 2 seconds before disappearing

Self.viewable: total number of frames the box will be viewable for (2 seconds * 120 frames per second for our game)

List of Errors

- “There is nothing to drop” if the user presses “f” (drop) while not holding anything
- “You can only hold one item at a time” if the user presses “d” (pick up) while already holding an item
- “You can’t carry more than 5 items” if the user presses “d” while already possessing 5 items in their backpack
- “There is nothing to pick up” if the user presses “d” while not touching an object they can pick up
- “There is nothing to place in your backpack” if the user presses “e” (place in backpack) while not holding anything
- “There is nothing in your backpack” if the user presses “r” (cycle through backpack) while nothing is in their backpack
- “You can only turn a flashlight on/off” if the user presses “q” (toggle flashlight) while holding an object other than a flashlight
- “You can only dig with a shovel” if the user extends an object that isn’t a shovel at a plant (i.e., if they try to dig with a different holdable item)
- “Wrong key. Hint: door and key colors should match” if the user tries to unlock a door with a key that doesn’t match
- “You can only open a door with a key” if the user extends an object that isn’t a key at a door (i.e., if they try to unlock the door with a different holdable item)
- “Find and turn on the flashlight to see in the labyrinth” if the user enters the labyrinth without a turned on flashlight
- “You can’t kill with the {key/flashlight/gem}” if the user extends the key, flashlight, or gem at a monster (i.e., tries to do damage with these items)
- “You need {1/2/3/4/5} more gems to open the door” if the user hasn’t placed all 5 gems into the space before the last door during the last level

ErrorBox(TextBox)

def __init__(self, x, y, width, height, text_size=25) ● Initiates the error pop-up texts
def update_text(self, new_text) ● Updates text based on the parameter new_text
def draw(self, screen, game) ● Display the text on the screen




Holdable


- Controls all the Holdable objects throughout the game, including
  - Extending motion when in use
  - Detecting touching/collision
  - Backpack usage
    - Picking up, dropping, looping through, placing in backpack, taking out, etc.
- Created by: Joy (Weapons and backpack/carrying item functionality), Spencer (Key, Potion, Gem)


Variables

Self.images: list of images for each frame of the holdable object; there is 1 frame for each direction the character faces (up, down, right, or left)

Self.width & self.height: width and height of the holdable item

Self.x & Self.y: x and y positions of the holdable object when held

Self.x_bg & Self.y_bg: x and y positions of the holdable object on the background (when NOT held)

Self.frame: which frame image should be used (direction the object is facing)

Self.walk_over: whether you are able to walk over the item, set True by default

Self.wielder: who is in control of the item

Wielder.held_item: the item held by the wielder

Self.loc: keeps track of where the item is, either: "hands", "backpack", or "ground"

Self.nick: internal name of the item, used for dictionary key

Self.times_picked_up: number of times the item has been picked up

List of Holdable Subclasses

01

Key(Holdable)

def __init__(self,wielder,loc,id,x_bg=0, y_bg=0)
- Initiates a Key
- id: corresponds to the door the key opens**

**contains additional variable self.id

02

Piece(Holdable)

def __init__(self,wielder,loc,x_bg=0, y_bg=0)
- Initiates a Piece object (a gem)

def in_space(self, x_right, x_left, y_top, y_bottom)
- Checks if the gem is in a particular area, marked by the input parameters (x_right, x_left, y_top, y_bottom)

03

Potion(Holdable)

def __init__(self,wielder,loc,x_bg=0,y_bg=0)
- Initiates a Potion object

def use_potion(self, player)
- Uses the potion and calls the become_immortal(self, frames=32, delay=25) function to end the game

Holdable

def __init__(self,wielder,loc,nick,sfx=default_sfx, x_bg=0, y_bg=0, defaultImages=True)
Initiates a weapon of Holdable class

- Wielder: who is holding/controlling the weapon (None if not being held)
- loc: location of the weapon relative to the player (e.g., ‘inhand’, ‘ground’)
- nick: nickname for the weapon
- x_bg, y_bg: x and y positions of item when not being held
- defaultImages = a boolean that checks how to retrieve images from the folder (e.g., for the flashlight there are two image sets, but for most there aren’t)

def detect_collision(self, other)

- Detects a collision between the weapon and another object/sprite (other)

Holdable

def draw(self, background, extend=False)

- draws/place the item on the screen
- if the player is holding the item, positions the item with reference to the player and makes sure that the weapon appears in the player's hand
- if the item is on the ground (not held by the player), draws the item in the background

def extend(self, background)

- extends the item so that it is 10 pixels further from the hands of the player when the item is in use

def pick_up(self, background, player)

- picks the item up from the ground and places it into the player's hands
- adds the item into the list of items the player can loop through and hold (the backpack)

Holdable

def drop(self, background, player)
- drops the item from the player's hands onto the ground (i.e., into the background)
- removes the item from the list of items that the player can loop through and hold (the backpack)

def place_in_backpack(self, player)
- removes the item from the player's hands and adds it into the backpack so that the player can carry the item
- the item is kept in the list of weapons the player can loop through and hold (backpack)

def select_from_backpack(self, player)
- Takes the weapon out of the backpack and places it into the player's hands
- the item is still in the list of weapons the player can loop through and hold (backpack)

def touching(self, other, background)
- Checks whether the weapon is touching a particular object (other)
- If yes, returns True; if not, returns False

Weapon(Holdable)

def __init__(self,wielder,loc,nick, sfx=default_sfx, x_bg=0, y_bg=0, defaultImages=True)

- Initiates a Weapon object, subclass of Holdable

Holdable objects that are also Weapons (subclass)
Created by: Joy

List of Weapon(Holdable) Subclasses

01

Sword

02

Shovel

03

Flashlight

04

Flame_Thrower

05

Trident

06

Boxing_Glove

Shovel(Weapon)

def __init__(self,wielder,loc, x_bg=0, y_bg=0) ● Initiates a Shovel
def dig(self, plant, background) ● Lets the player ‘dig’ the plant to gain health points

Flashlight(Weapon)

def __init__(self,wielder,loc,x_bg=0, y_bg=0) • Initiates a Flashlight
def change_state(self) • Opens and closes the flashlight • Calls the turn_on(self) and turn_off(self) functions based on the value of self.state (off if True, on if False)
def turn_on(self) • Turns on flashlight and sets self.state = True
def turn_off(self) • Turns off flashlight and sets self.state = False

Variables:

- Self.images_on; flashlight sprites - on
- Self.images_off; flashlight sprites - off
- Self.images: flashlight sprites in use
- Self.state: tracks whether the flash light is on or off (boolean)

General Division of Labor

Spencer

- Drawing/Sprites
- Background and Doors classes
- Non-Weapon Holdable classes (Key, Gem, Potion)
- Player class
- Menu and menu items
- Game class functions pertaining to above items

Amanda

- Written storyline
- Interactable background objects (Chest, Plant)
- Health Bars
- Monster class
- Most text classes (TextBox, ImageBox, Story)
- Game class functions pertaining to above items

Jue

- Documentation/Slideshow
- Holdable functions pertaining to carrying items/backpack
- Weapon class
- ErrorBox class
- Button and BackgroundButton classes
- Game class functions pertaining to above items

Thank You!