COMPUTER SCIENCE PROJECT
FOR THE YEAR

-2023-2024-

--Submitted by--

Joseph Chacko
Vasudev Dinesh
Sooreya Narayanan MS

<INDEX>

<Acknowledgments>

I want to express my profound gratitude to God for providing me with the strength and inspiration to embark on this journey. To my dedicated teachers, who imparted knowledge and guidance, I owe a debt of thanks for nurturing my skills. To my supportive friends, you've been my rock and a source of unwavering encouragement. I'm also deeply appreciative of those who provided opportunities that allowed me to pursue my passion. Each of you has played an integral role in the creation of ColdZap, and I'm sincerely thankful for your support.

<Project Description>

ColdZap is an intriguing gaming endeavour meticulously crafted in Python, leveraging the potent pygame library. It falls within the captivating realm of dungeon explorer-style games, where players embark on a thrilling journey through intricately designed levels. However, this game diverges from the mundane; it requires players to not only navigate these treacherous domains but also eliminate all adversaries in their path to unlock access to subsequent levels.

What sets ColdZap apart is its emphasis on strategic thinking and environmental interaction. Success is not solely determined by brute force; instead, players must employ wit and ingenuity to conquer challenges swiftly. Each level presents a puzzle to be solved, requiring players to exploit the surroundings and devise efficient solutions. It's a mental exercise as much as a test of reflexes.

<Files>

```
|Assets [All asset dependencies needed by the game.]
|-> audio
|--> ...
|
|-> fonts
|--> ...
|
|-> images
|--> ...
|
|GameData [All data related to the game.]
|-> Levels
|--> level0.json
|--> level1.json
|--> level2.json
|--> level3.json
|--> level4.json
|
|-> highscores.json [High Scores]
|-> saves.json [The Savefile.]
|-> settings.json [User Preferences.]
|
|utils [This is a custom library.(Utilities)]
|-> __init__.py
|-> blocks.py
|-> bullet.py
|-> enemy.py
|-> player.py
|-> txt_button.py
|
|drawing_functions.py [Functions dealing with rendering]
|levelCreate.py [A GUI for level creation.]
|main.py [Entry point.]
```

<Functions &Classes>

<Drawing>

```
def draw_bg(surface):
    surface.fill("#baf9ff")
    pg.draw.rect(surface, "#a4eeff", (0, 0, 350 *
                    SCALE * SCALE, 450 * SCALE), 5)
    for x, y in [(i, j) for i in range(7) for j in
                    range(9) if (i + j) % 2]:
        pg.draw.rect(
            surface, "#a4eeff", (50 * SCALE * x, 50 *
            SCALE * y, 50 * SCALE, 50 * SCALE))
```

>> This function takes care of rendering the checker board background seen behind the game levels.

```
def draw_menu_bg(surface):
    surface.fill("#d8fcff")
    for x, y in [(i, j) for i in range(7) for j in
                    range(10) if (i + j) % 2]:
        pg.draw.rect(
            surface, "#bef3ff", (50 * SCALE * x, 50 *
            SCALE * y, 50 * SCALE, 50 * SCALE))
```

>> The same function but for the menu (There are subtle differences in requirements.).

```
def draw_txt(surface, txt, x, y, color, font,
align="center"):
    label = font.render(txt, True, color)
    rect = (
        label.get_rect(center=(x, y))
        if align == "center"
        else label.get_rect(midleft=(x, y))
    )
    surface.blit(label, rect)
```

>> Used To render text onto the screen.

```
def draw_ui(surface, font: pg.font.Font, level,
score, lives):
    score = "Score: " + str(score)
    level = "Level: " + str(level)
    pg.draw.rect(surface, "#baf9ff", (0, 450 * SCALE,
                 350 * SCALE, 50 * SCALE))
    draw_txt(surface, str(level), 210 * SCALE, 465 *
             SCALE, (0, 0, 0), font, "")
    draw_txt(surface, str(score), 210 * SCALE, 490 *
             SCALE, (0, 0, 0), font, "")

    for i in range(lives):
        rect = pg.Rect(60 * SCALE + 25 * SCALE * i,
                       470 * SCALE, 20 * SCALE, 20 *
                       SCALE)
        pg.draw.rect(surface, (255, 0, 0), rect, 0,
                                  5)
```

>> Draws the UI elements onto the screen.

```
def fade_to(surface, color, duration):
    surf = pg.Surface((700 * SCALE, 500 * SCALE))
    surf.fill(color)
    for i in range(60):
        surf.set_alpha(int(17))
        surface.blit(surf, (0, 0))
        pg.display.flip()
        pg.time.delay(int(duration * 1000 / 60))
```

>> Used to give a “fade” transition effect.

<Gameplay>

```
def displayBullets(screen):
    for i in Bulletlist:
        if i.update(screen):
            Bulletlist.remove(i)
```

>> Function to iterate over and display the bullets (Both player and enemy). Also handles bullet removal when out of bounds

```
def update_enemies(screen):
    hit = False
    for i in Enemylist:
        if i.update(screen):
            hit = True
    return hit
```

>> Function to iterate over all the enemies and update and draw them

```
def update_collidables(screen):
    for i in Collidable_list:
        i.update(screen)
```

>> Function to iterate over and update all collidable entities.

<Classes>

```
class Wall:
    """
    Class for the walls in the game.

    :param pos: The position of the wall.
    """

    def __init__(self, pos):
        x, y = pos
        self.image = pg.image.load(f"assets/images/{SCALE}x/wall2.png")
        self.pos = Vector2(x, y)
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.mask = pg.mask.from_surface(self.image)
        Collidable_list.append(self)

    def update(self, surface):
        for i in Bulletlist:
            if self.mask.overlap(
                i.mask,
                (
                    int(i.pos.x - self.pos.x * 50 * SCALE),
                    int(i.pos.y - self.pos.y * 50 * SCALE),
                ),
            ):
                Bulletlist.remove(i)

        surface.blit(self.image, self.rect)
```

>> This class is the blueprint of the walls that make up levels.

>> It contains methods to update and draw the Wall.

```
class Pit:
    """
    Class for the pits in the game.

    :param pos: The position of the pit.
    """

    def __init__(self, pos):
        x, y = pos
        self.image = pg.image.load(f"assets/images/{SCALE}x/pit2.png")
        self.pos = Vector2(x, y)
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.mask = pg.mask.from_surface(self.image)
        Collidable_list.append(self)

    def update(self, surface):
        surface.blit(self.image, self.rect)
```

>> This class is the blueprint of the pits that make up levels.

>> It contains methods to update and draw the Pit.

```
class Bullet:
    """
    Class for the bullets in the game.

    :param pos: The position of the bullet.
    :param vel: The velocity of the bullet. Vector stores both direction
                and speed.
    :param type: The type of the bullet.
    """

    def __init__(self, pos, vel, type="red"):
        self.type = type
        self.image = pg.image.load(f"assets/images/{SCALE}x/{type}-
bullet.png")
        self.pos = pos * 50 * SCALE + Vector2(25, 25) * SCALE
        self.vel = vel * SCALE
        self.dir = vel.angle_to(Vector2(0, -1))
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.mask = pg.mask.from_surface(self.image)

    def update(self, surface):
        tempimage = pg.transform.rotate(self.image, self.dir)
        self.pos += self.vel
        self.rect.center = self.pos
        surface.blit(tempimage, self.rect)

        if not (0 < self.pos.x < 7 * SCALE * 50 and 0 < self.pos.y < 9 *
                SCALE * 50):
            return True
```

>> This class is the blueprint of the bullets that make shoooting possible

>> It contains methods to update and draw the Bullets

```
class Enemy:
    """
    Class for the enemies in the game.

    :param type: The type of the enemy.
    :param positions: The positions of the enemy.
    """

    def __init__(
        self,
        type: str,
        positions: list[list[int, int]],
    ):
        self.type = type
        self.positions = positions + positions[::-1]

        self.image = pg.image.load(f"assets/images/{SCALE}x/{self.type}.png")
        self.rect = self.image.get_rect()

        self.counter = 0

        self.mask = pg.mask.from_surface(self.image)

        self.pos = Vector2(self.positions[self.counter])
        self.tgt = Vector2(self.positions[1])

        self.health = 5

        Enemylist.append(self)

    def update(self, surface):
        if self.counter > 120:
            self.counter = 0
            Bulletlist.append(
                Bullet(self.pos, (player.PlayerPos - self.pos).normalize() * 2)
            )

        self.counter += 1

        if (self.tgt - self.pos).length() < 0.05:
            self.pos = self.tgt
            self.positions.append(self.positions.pop(0))
            self.tgt = Vector2(self.positions[0])
        self.pos = self.pos + (self.tgt - self.pos) * 0.1
        self.rect.center = self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        surface.blit(self.image, self.rect)

        self.collision()
```

```
        if self.health <= 0:
            try:
                Enemylist.remove(self)
                return True
            except Exception as e:
                pass

    def collision(self):
        for i in Bulletlist:
            if i.type == "blue":
                if self.mask.overlap(
                    i.mask,
                    (
                        int(i.pos.x - self.pos.x * 50 * SCALE),
                        int(i.pos.y - self.pos.y * 50 * SCALE),
                    ),
                ):
                    Bulletlist.remove(i)
                    self.health -= 1
```

>> The core class for all the enemies in the game.
>> It handles enemy behaviour, updation and drawing.

```python
class Player:

    """
    INITALIZE THE PLAYER
    Syntax: Player(x, y, health)

    :param x: x position of the player
    :param y: y position of the player
    :param health: health of the player
    """

    def __init__(self, x=3, y=8, health=5):
        global PlayerPos
        self.image = pg.image.load(f"assets/images/{SCALE}x/player.png")
        self.pos = Vector2(x, y)
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.moving = False
        self.vel = Vector2(0, 0)
        self.target = self.pos
        self.targets = []
        self.mask = pg.mask.from_surface(self.image)
        self.health = health
        self.counter = 0
        self.hit = False

    def move(self, direction):
        match direction:
            case "up":
                self.targets.append(Vector2(0, -1))
            case "down":
                self.targets.append(Vector2(0, 1))
            case "left":
                self.targets.append(Vector2(-1, 0))
            case "right":
                self.targets.append(Vector2(1, 0))

    def is_walkable(self, direction, pos=None):
        if pos == None:
            pos = self.pos

        for i in utils.Collidable_list:
            if i.pos == pos + direction:
                return False
        return True

    def update(self, surface):
        global PlayerPos
```

```
        if self.moving:
            if (self.target - self.pos).length() < 0.05:
                self.pos = self.target
                self.moving = False
            self.pos = self.pos + (self.target - self.pos) * 0.3
        else:
            if len(self.targets) != 0:
                direction = self.targets.pop(0)
                target = self.pos + direction
                if (
                    -1 < target.x < 7
                    and -1 < target.y < 9
                    and self.is_walkable(direction)
                ):
                    self.target = target
                    self.moving = True
            self.tgt = self.pos
        PlayerPos = self.pos
        self.rect.center = self.pos * 50 * SCALE + Vector2(25, 25) * SCALE

        self.collision()
        self.shoot_stuff()

        surface.blit(self.image, self.rect)

    def shoot_stuff(self):
        try:
            if self.counter > 10:
                for enemy in utils.Enemylist:
                    if self.pos.x == enemy.tgt.x or self.pos.y == enemy.tgt.y:
                        utils.Bulletlist.append(
                            utils.Bullet(
                                self.pos,
                                (enemy.tgt - self.pos).normalize() * 20,
                                "blue",
                            )
                        )
                        self.counter = 0
            self.counter += 1
        except:
            pass

    def collision(self):
        for i in utils.Bulletlist:
            if i.type == "red":
                if self.mask.overlap(
                    i.mask,
                    (
                        int(i.pos.x - self.pos.x * 50 * SCALE),
                        int(i.pos.y - self.pos.y * 50 * SCALE),
                    ),
```

```
            ):
                self.hit = True
                utils.Bulletlist.remove(i)
                self.health -= 1
```

>> The core class for hangling the player in the game.
>> It handles input, updation, shooting and drawing.

```
class TxtButton:
    """
    INITALIZE THE BUTTON
    Syntax: TxtButton(x, y, txt, color, font)

    :param x: x position of the button
    :param y: y position of the button
    :param txt: text to be displayed on the button
    :param color: color of the text
    :param font: font of the text
    """

    def __init__(self, x, y, txt, color, font):
        self.txt = txt
        self.label = font.render(txt, True, color)
        self.rect = self.label.get_rect(center=(x, y))
        self.bgrect = self.rect.inflate(10, 10)
        self.clicked = False
        self.color = color
        self.font = font

    def update(self, surface, pos):
        action = False

        self.label = self.font.render(self.txt, True, self.color)
        self.bgrect = self.rect.inflate(10, 10)
        self.rect = self.label.get_rect(center=self.rect.center)

        mp = pg.mouse.get_pos()
        if self.rect.collidepoint(mp):
            pg.draw.rect(surface, "#dddddd", self.bgrect)
        if self.rect.collidepoint(pos):
            action = True
        surface.blit(self.label, self.rect)
        return action
```

>> All of the buttons on the menus are instances of this class.
>> It handles input, updation and drawing.

<Source Code>

<main.py>

```
import pygame as pg
import sys
from pygame.math import Vector2
import json

import drawing_functions as df
import utils

# ---INITIALISATION-----------------------------
from utils import SCALE

pg.init()
screen = pg.display.set_mode((int(350 * SCALE), int(500 * SCALE)))
pg.display.set_caption("ColdZap")

# ---VARIABLES----------------------------------
quit = False

clock = pg.time.Clock()
# ---FONTS--------------------------------------
Comfortaa = pg.font.Font("assets/fonts/Comfortaa.ttf", int(60 * SCALE))
Comfortaa_small = pg.font.Font("assets/fonts/Comfortaa.ttf", int(20 * SCALE))
# ---MUSIC--------------------------------------
music_playing = False
current_song = "Nothing"

intro_sound = pg.mixer.Sound("assets/audio/Level-Intro.wav")


# ---FUNCTIONS----------------------------------
def load_settings():
    global music_playing, current_song

    with open("GameData/settings.json") as f:
        settings = json.load(f)
        if settings["music"] == 0:
            music_playing = False
            current_song = "Nothing"
        elif settings["music"] == 1:
            music_playing = True
```

```
            current_song = "Astra"
            pg.mixer.music.load("assets/audio/Level.mp3")
        elif settings["music"] == 2:
            music_playing = True
            current_song = "Supert"
            pg.mixer.music.load("assets/audio/Supert.mp3")
    if music_playing:
        pg.mixer.music.play(-1)
    else:
        pg.mixer.music.stop()


load_settings()


# ---SCREEN-FUNCTIONS-------------------------
def main(saved=False):
    utils.Bulletlist.clear()
    utils.Enemylist.clear()
    utils.Collidable_list.clear()

    def wincheck():
        if not utils.Enemylist:
            return True
        else:
            return False

    if saved:
        with open("Gamedata/saves.json") as f:
            save = json.load(f)
            level = save["levelId"]
            score = save["score"]
            lives = save["lives"]
    else:
        level = 0
        score = 0
        lives = 5

        with open("Gamedata/saves.json", "w") as f:
            json.dump({"levelId": level, "score": score, "lives": lives},
                      f,indent=4)

    with open(f"Gamedata/Levels/Level{level}.json") as f:
        level_data = json.load(f)
    for i in level_data["enemies"]:
        utils.Enemy(
            level_data["enemies"][i]["type"],
            level_data["enemies"][i]["positions"],
        )

    for i in level_data["wallPositions"]:
```

```
        utils.Wall(i)

    for i in level_data["pitPositions"]:
        utils.Pit(i)

    startpos = level_data["playerStartPosition"]

    player = utils.Player(startpos[0], startpos[1], lives)
    global quit

    def event_handler():
        for event in pg.event.get():
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_UP or event.key == pg.K_w:
                    player.move("up")
                elif event.key == pg.K_DOWN or event.key == pg.K_s:
                    player.move("down")
                elif event.key == pg.K_LEFT or event.key == pg.K_a:
                    player.move("left")
                elif event.key == pg.K_RIGHT or event.key == pg.K_d:
                    player.move("right")
                elif event.key == pg.K_ESCAPE or event.key == pg.K_q:
                    return True

    back_button = utils.TxtButton(
        20 * SCALE, 480 * SCALE, "<=", (0, 0, 0), Comfortaa_small
    )
    pg.mixer.music.stop()
    intro_sound.play()
    pg.mixer.music.load("assets/audio/Level-Theme.wav")
    pg.mixer.music.play(-1)
    while not quit:
        clock.tick(60)
        quit = (
            pg.event.get(pg.QUIT) or event_handler()
        )  # quit if window is closed or event_handler returns True

        df.draw_bg(screen)  # draw background

        utils.update_collidables(screen)

        utils.displayBullets(screen)

        player.update(screen)  # update player

        if utils.update_enemies(screen):
            score += 10  # update enemies and add 10 to score if enemy is killed

        df.draw_ui(screen, Comfortaa_small, level, score, player.health)  # draw ui

        if back_button.update(
```

```
            screen, pg.mouse.get_pos() if pg.mouse.get_pressed()[0] else (0, 0)
        ):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return menu, ()

        if player.health == 0:
            with open("Gamedata/saves.json", "w") as f:
                json.dump({"levelId": 0, "score": 0, "lives": 5}, f,indent=4)

            with open("Gamedata/highscores.json", "r") as f:
                highscores = json.load(f)
            if score > int(highscores["highscore"]):
                with open("Gamedata/highscores.json", "w") as f:
                    json.dump({"highscore": str(score)}, f,indent=4)

            df.fade_to(screen, (0, 0, 0), 0.15)
            return you_died, ()

        pg.display.flip()

        if wincheck():
            df.fade_to(screen, (0, 0, 0), 0.5)
            if level + 1 == 5:
                raise NotImplementedError(f"Level {level+1} not implemented yet")
            else:
                with open("GameData/saves.json", "w") as f:
                    print(level + 1)
                    json.dump(
                        {"levelId": level + 1, "score": score, "lives":
                         player.health},
                        f,
                        indent=4
                    )
            with open("Gamedata/highscores.json", "r") as f:
                highscores = json.load(f)
            if score > int(highscores["highscore"]):
                with open("Gamedata/highscores.json", "w") as f:
                    json.dump({"highscore": str(score)}, f,indent=4)

            return main, (True,)

    pg.quit()
    sys.exit()


def you_died():
    menu_button = utils.TxtButton(
        175 * SCALE, 450 * SCALE, "Back to menu", (0, 0, 0), Comfortaa_small
    )

    label = Comfortaa.render("You died", True, (0, 0, 0))
```

```
    label_rect = label.get_rect(center=(175 * SCALE, 100 * SCALE))

    def event_handler():
        for event in pg.event.get():
            if event.type == pg.QUIT:
                return True, ()
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_ESCAPE or event.key == pg.K_q:
                    return True, ()
            if event.type == pg.MOUSEBUTTONDOWN:
                return False, event.pos
        return False, ()

    quit = False
    while not quit:
        clock.tick(60)
        ev = event_handler()

        quit = ev[0]

        df.draw_menu_bg(screen)  # draw background
        mouse_pos = ev[1] if ev[1] else (0, 0)

        screen.blit(label, label_rect)

        if menu_button.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return menu, ()

        pg.display.flip()

    pg.quit()
    sys.exit()


# ------------------------------------------
def menu():
    global quit

    load_settings()
    def event_handler():
        for event in pg.event.get():
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_ESCAPE or event.key == pg.K_q:
                    return True
                elif event.key == pg.K_RETURN:
                    return False

    new_game = utils.TxtButton(
        175 * SCALE, 200 * SCALE, "New Game", (0, 0, 0), Comfortaa_small
    )
```

```
    load_game = utils.TxtButton(
        175 * SCALE, 250 * SCALE, "Load Game", (0, 0, 0), Comfortaa_small
    )
    view_highscore = utils.TxtButton(
        175 * SCALE, 300 * SCALE, "Highscores", (0, 0, 0), Comfortaa_small
    )
    settings_button = utils.TxtButton(
        175 * SCALE, 350 * SCALE, "Settings", (0, 0, 0), Comfortaa_small
    )
    quit_game = utils.TxtButton(
        175 * SCALE, 400 * SCALE, "Quit", (0, 0, 0), Comfortaa_small
    )

    while not quit:
        clock.tick(60)

        quit = (
            pg.event.get(pg.QUIT) or event_handler()
        )  # quit if window is closed or event_handler returns True

        df.draw_menu_bg(screen)  # draw background
        mouse_pos = (
            pg.mouse.get_pos() if pg.mouse.get_pressed()[0] else (0, 0)
        )  # get mouse position if mouse is pressed, else (0,0)

        df.draw_txt(screen, "ColdZap", 175 * SCALE, 100 * SCALE, (0, 0, 0),
Comfortaa)

        if quit_game.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            break
        if load_game.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return main, (True,)
        if view_highscore.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return highscore, ()
        if settings_button.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return settings, ()
        if new_game.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return main, (False,)

        pg.display.flip()

    pg.quit()
    sys.exit()


# ---------------------------------------------
```

```
def highscore():
    with open("Gamedata/highscores.json") as f:
        highscore = json.load(f)

    labels = [
        Comfortaa_small.render("Current Highscore", True, (0, 0, 0)),
        Comfortaa.render(str(highscore["highscore"]), True, (0, 0, 0)),
    ]

    label_rects = [
        labels[0].get_rect(center=(175 * SCALE, 200 * SCALE)),
        labels[1].get_rect(center=(175 * SCALE, 250 * SCALE)),
    ]

    menu_button = utils.TxtButton(
        175 * SCALE, 450 * SCALE, "Back to menu", (0, 0, 0), Comfortaa_small
    )

    def event_handler():
        for event in pg.event.get():
            if event.type == pg.QUIT:
                return True, ()
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_ESCAPE or event.key == pg.K_q:
                    return True, ()
            if event.type == pg.MOUSEBUTTONDOWN:
                return False, event.pos
        return False, ()

    quit = False
    while not quit:
        clock.tick(60)
        ev = event_handler()

        quit = ev[0]

        df.draw_menu_bg(screen)  # draw background

        mouse_pos = ev[1] if ev[1] else (0, 0)

        for i in range(len(labels)):
            screen.blit(labels[i], label_rects[i])

        if menu_button.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return menu, ()

        pg.display.flip()

    pg.quit()
    sys.exit()
```

```
# ------------------------------------------
def settings():
    global quit, current_song

    def event_handler():
        for event in pg.event.get():
            if event.type == pg.QUIT:
                return True, ()
            if event.type == pg.KEYDOWN:
                if event.key == pg.K_ESCAPE or event.key == pg.K_q:
                    return True, ()
            if event.type == pg.MOUSEBUTTONDOWN:
                return False, event.pos
        return False, ()

    music_button = utils.TxtButton(
        175 * SCALE, 100 * SCALE, f"Music : {current_song}", (0, 0, 0),
Comfortaa_small
    )
    menu_button = utils.TxtButton(
        175 * SCALE, 150 * SCALE, "Back to menu", (0, 0, 0), Comfortaa_small
    )

    while not quit:
        clock.tick(60)
        ev = event_handler()

        quit = ev[0]  # quit if window is closed or event_handler returns True

        df.draw_menu_bg(screen)  # draw background
        mouse_pos = (
            ev[1] if ev[1] else (0, 0)
        )  # get mouse position if mouse is pressed, else (0,0)

        if music_button.update(screen, mouse_pos):
            # cycle through songs
            if current_song == "Nothing":
                current_song = "Astra"
                with open("GameData/settings.json") as f:
                    settings = json.load(f)
                    settings["music"] = 1
                with open("GameData/settings.json", "w") as f:
                    json.dump(settings, f,indent=4)
            elif current_song == "Astra":
                current_song = "Supert"
                with open("GameData/settings.json") as f:
                    settings = json.load(f)
                    settings["music"] = 2
                with open("GameData/settings.json", "w") as f:
```

```
                    json.dump(settings, f,indent=4)
            elif current_song == "Supert":
                current_song = "Nothing"

                with open("GameData/settings.json") as f:
                    settings = json.load(f)
                    settings["music"] = 0
                with open("GameData/settings.json", "w") as f:
                    json.dump(settings, f,indent=4)

            load_settings()
            music_button.txt = f"Music : {current_song}"

        if menu_button.update(screen, mouse_pos):
            df.fade_to(screen, (0, 0, 0), 0.15)
            return menu, ()

        pg.display.flip()

    pg.quit()
    sys.exit()


# ----------------------------------------------

# ---ENTRY-POINT--------------------------------
if __name__ == "__main__":
    active_screen = menu
    args = ()
    while True:
        active_screen, args = active_screen(*args)
# ----------------------------------------------
```

<drawing_functions.py>

```
import pygame as pg

# ---VARIABLES-----------------------------------
from utils import SCALE

# -----------------------------------------------
# ---FUNCTIONS-----------------------------------
def draw_bg(surface):
    surface.fill("#baf9ff")
    pg.draw.rect(surface, "#a4eeff", (0, 0, 350 * SCALE * SCALE, 450 * SCALE), 5)
    for x, y in [(i, j) for i in range(7) for j in range(9) if (i + j) % 2]:
        pg.draw.rect(
            surface, "#a4eeff", (50 * SCALE * x, 50 * SCALE * y, 50 * SCALE, 50 *
               SCALE)
        )


# -----------------------------------------------
def draw_menu_bg(surface):
    surface.fill("#d8fcff")
    for x, y in [(i, j) for i in range(7) for j in range(10) if (i + j) % 2]:
        pg.draw.rect(
            surface, "#bef3ff", (50 * SCALE * x, 50 * SCALE * y, 50 * SCALE, 50 *
               SCALE)
        )


# -----------------------------------------------
def draw_txt(surface, txt, x, y, color, font, align="center"):
    label = font.render(txt, True, color)
    rect = (
        label.get_rect(center=(x, y))
        if align == "center"
        else label.get_rect(midleft=(x, y))
    )
    surface.blit(label, rect)


# -----------------------------------------------
def fade_to(surface, color, duration):
    surf = pg.Surface((700 * SCALE, 500 * SCALE))
    surf.fill(color)
    for i in range(60):
        surf.set_alpha(int(17))
        surface.blit(surf, (0, 0))
        pg.display.flip()
```

```
        pg.time.delay(int(duration * 1000 / 60))


# ------------------------------------------------
def draw_ui(surface, font: pg.font.Font, level, score, lives):
    score = "Score: " + str(score)
    level = "Level: " + str(level)
    pg.draw.rect(surface, "#baf9ff", (0, 450 * SCALE, 350 * SCALE, 50 * SCALE))
    draw_txt(surface, str(level), 210 * SCALE, 465 * SCALE, (0, 0, 0), font, "")
    draw_txt(surface, str(score), 210 * SCALE, 490 * SCALE, (0, 0, 0), font, "")

    for i in range(lives):
        rect = pg.Rect(60 * SCALE + 25 * SCALE * i, 470 * SCALE, 20 * SCALE, 20 *
                       SCALE)
        pg.draw.rect(surface, (255, 0, 0), rect, 0, 5)


# ------------------------------------------------
```

<levelCreate.py>

```
import pygame as pg
import json

"""
A simple utility to create levels for the game.
"""

LEVEL_ID = 4  # Change this to the level you want to edit
IMPLEMENTED_LEVELS = [1, 2, 3, 4]  # Add the levels you have implemented here

from utils import SCALE

COLORS = [
    "white",
    "red",
    "green",
    "blue",
    "yellow",
    "orange",
    "purple",
    "pink",
    "brown",
    "cyan",
]


pg.init()
screen = pg.display.set_mode((int(350 * SCALE), int(450 * SCALE)))
font = pg.font.Font("assets/fonts/Comfortaa.ttf", 20)
bfont = pg.font.Font("assets/fonts/Comfortaa.ttf", 40)


def load_tiles(level_id):
    data = json.load(open("Gamedata/Levels/level" + str(level_id) + ".json"))
    tiles = [Tile(x, y) for x in range(7) for y in range(9)]

    for tile in tiles:
        if [tile.x, tile.y] in data["wallPositions"]:
            tile.txt = "W"
            tile.value = REFERENCES["W"](tile.x, tile.y)
        elif [tile.x, tile.y] in data["pitPositions"]:
            tile.txt = "P"
            tile.value = REFERENCES["P"](tile.x, tile.y)
        elif [tile.x, tile.y] == data["playerStartPosition"]:
            tile.txt = "O"
            tile.value = REFERENCES["O"](tile.x, tile.y)
```

```
        for enemy in data["enemies"]:
            enemy = data["enemies"][enemy]
            enemy_positions = enemy["positions"]

            if enemy_positions[0] == [tile.x, tile.y]:
                tile.txt = "E" if enemy["type"] == "glider" else "AE"
                tile.value = REFERENCES[tile.txt](tile.x, tile.y, enemy_positions)

    return tiles


def debug(txt):
    screen.blit(font.render(txt, True, "white", "black"), (0, 0))


def encode_into(filename, tiles):
    data = {
        "levelId": 1,
        "playerStartPosition": [3, 0],
        "pitPositions": [],
        "wallPositions": [],
        "enemies": {},
    }

    for tile in tiles:
        if tile.txt == "W":
            data["wallPositions"].append([tile.value.x, tile.value.y])
        elif tile.txt == "P":
            data["pitPositions"].append([tile.value.x, tile.value.y])
        elif tile.txt == "O":
            data["playerStartPosition"] = [tile.value.x, tile.value.y]
        elif tile.txt == "E":
            data["enemies"]["enemy" + str(len(data["enemies"]) + 1)] = {
                "type": "glider",
                "positions": tile.value.positions,
            }
        elif tile.txt == "AE":
            data["enemies"]["enemy" + str(len(data["enemies"]) + 1)] = {
                "type": "glider-advanced",
                "positions": tile.value.positions,
            }

    json.dump(data, open("Gamedata/Levels/" + filename, "w"), indent=4)


class WallTile:
    def __init__(self, x, y):
        self.x = x
        self.y = y


class PitTile:
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y


class EnemyTile:
    def __init__(self, x, y, positions):
        self.x = x
        self.y = y
        self.positions = positions
        self.color = COLORS.pop()
        print(positions)


class PlayerTile:
    def __init__(self, x, y):
        self.x = x
        self.y = y


class GoalTile:
    def __init__(self, x, y):
        self.x = x
        self.y = y


class AdvancedEnemyTile:
    """
    This class is not used in the game, but it is here for future use.
    """

    def __init__(self, x, y, positions):
        self.x = x
        self.y = y
        self.positions = positions


class Tile:
    """
    A tile is a single square on the level editor.
    """

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.txt = ""
        self.rect = pg.Rect(
            10 + self.x * 40 * SCALE, 70 + self.y * 40 * SCALE, 40 * SCALE, 40 *
               SCALE
        )
        self.value = None
```

```
    def update(self):
        hover = True if self.rect.collidepoint(pg.mouse.get_pos()) else False
        if hover:
            pg.draw.rect(screen, "#222222", self.rect)
        if self.txt != "":
            label = bfont.render(self.txt, True, "white")
            labelrect = label.get_rect(center=self.rect.center)
            screen.blit(label, labelrect)
        pg.draw.rect(screen, "white", self.rect, 1)
        if self.txt == "E" or self.txt == "AE":
            enemy_paths.append([self.value.positions, self.value.color])

    def check_click(self):
        if self.rect.collidepoint(pg.mouse.get_pos()):
            global currently_selected
            if mode == "enemy" or mode == "advanced_enemy":
                if [self.x, self.y] not in positions:
                    positions.append([self.x, self.y])
            elif currently_selected != None:
                if str(currently_selected) == "O":
                    for tile in TILES:
                        if tile.txt == "O":
                            tile.txt = ""
                            tile.value = None
                if self.txt == str(currently_selected):
                    self.txt = ""
                    self.value = None
                else:
                    self.txt = str(currently_selected)
                    self.value = REFERENCES[currently_selected.txt](self.x, self.y)


class TileSelector:
    """
    A tile selector is a single square on the right side of the level editor.
    It is used to select a tile to place.
    """

    def __init__(self, x, y, txt):
        self.x = x
        self.y = y
        self.txt = txt
        self.rect = pg.Rect(
            30 + self.x * 40 * SCALE, 70 + self.y * 45 * SCALE, 40 * SCALE, 40 *
               SCALE
        )

    def __str__(self):
        return self.txt

    def update(self):
```

```
        hover = True if self.rect.collidepoint(pg.mouse.get_pos()) else False
        if hover:
            pg.draw.rect(screen, "#222222", self.rect)
        pg.draw.rect(screen, "red", self.rect, 1)

        label = bfont.render(self.txt, True, "red")
        labelrect = label.get_rect(center=self.rect.center)
        screen.blit(label, labelrect)

    def check_click(self):
        if self.rect.collidepoint(pg.mouse.get_pos()):
            global currently_selected, mode, enemy_paths
            currently_selected = self
            if self.txt == "E" or self.txt == "AE":
                if mode == "edit":
                    mode = "enemy" if self.txt == "E" else "advanced_enemy"
                elif (mode == "enemy" or mode == "advanced_enemy") and
                      any(positions):
                    x, y = positions[-1]
                    for tile in TILES:
                        if tile.x == x and tile.y == y:
                            found_tile = tile
                            break

                    found_tile.txt = str(currently_selected)
                    found_tile.value = REFERENCES["E" if mode == "enemy" else
                                                  "AE"](
                        x, y, positions.copy()
                    )
                    positions.clear()
                    mode = "edit"
                    currently_selected = None
                else:
                    mode = "edit"
                    currently_selected = None


TILE_TYPES = ["W", "P", "O", "E", "AE", "G"]
REFERENCES = {
    "W": WallTile,
    "P": PitTile,
    "O": PlayerTile,
    "E": EnemyTile,
    "AE": AdvancedEnemyTile,  # This is not used in the game, but it is here for
                              # future use.
    "G": GoalTile,
}
TILES = (
    [Tile(x, y) for x in range(7) for y in range(9)]
    if LEVEL_ID not in IMPLEMENTED_LEVELS
    else load_tiles(LEVEL_ID)
```

```
)
TILE_SELECTORS = [
    TileSelector(7, y, txt)
    for y, txt in enumerate(TILE_TYPES)
    if txt not in ["AE", "G"]
]

currently_selected = None

mode = "edit"
positions = []
enemy_paths = []

while True:
    enemy_paths = []
    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()
            quit()
        if event.type == pg.MOUSEBUTTONDOWN:
            for tile in TILES + TILE_SELECTORS:
                tile.check_click()
        if event.type == pg.KEYDOWN:
            if event.key == pg.K_s:
                print("Saving...")
                encode_into("level" + str(LEVEL_ID) + ".json", TILES)
            if event.key == pg.K_ESCAPE:
                mode = "edit"
    screen.fill("black")
    for tile in TILES + TILE_SELECTORS:
        tile.update()

    for path, color in enemy_paths:
        pg.draw.lines(
            screen,
            color,
            False,
            [
                (x * 40 * SCALE + 10 + SCALE * 20, y * 40 * SCALE + 70 + SCALE *
                  20)
                for x, y in path
            ],
            2,
        )

    debug(
        f"Currently selected: {currently_selected} | Mode: {mode} | Positions:
            {str(positions)}"
    )

    pg.display.flip()
```

<utils/__init__py>

```
import pygame as pg

# ---VARIABLES-----------------------------------
SCALE = 1.7
Collidable_list = []
# -----------------------------------------------
# ---CLASSES-------------------------------------
from .txt_button import TxtButton
from .bullet import Bulletlist, Bullet
from .enemy import Enemylist, Enemy
from .player import Player
from .blocks import Wall, Pit


# -----------------------------------------------
# ---FUNCTIONS-----------------------------------
def displayBullets(screen):
    for i in Bulletlist:
        if i.update(screen):
            Bulletlist.remove(i)


# -----------------------------------------------
def update_enemies(screen):
    hit = False
    for i in Enemylist:
        if i.update(screen):
            hit = True
    return hit


# -----------------------------------------------
def update_collidables(screen):
    for i in Collidable_list:
        i.update(screen)


# -----------------------------------------------
```

<utils/blocks.py>

```
import pygame as pg
from pygame.math import Vector2
from utils import Bulletlist, Collidable_list, SCALE

"""
Contains all the block classes in the game.
"""


class Wall:
    """
    Class for the walls in the game.

    :param pos: The position of the wall.
    """

    def __init__(self, pos):
        x, y = pos
        self.image = pg.image.load(f"assets/images/{SCALE}x/wall2.png")
        self.pos = Vector2(x, y)
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.mask = pg.mask.from_surface(self.image)
        Collidable_list.append(self)

    def update(self, surface):
        for i in Bulletlist:
            if self.mask.overlap(
                i.mask,
                (
                    int(i.pos.x - self.pos.x * 50 * SCALE),
                    int(i.pos.y - self.pos.y * 50 * SCALE),
                ),
            ):
                Bulletlist.remove(i)

        surface.blit(self.image, self.rect)


class Pit:
    """
    Class for the pits in the game.

    :param pos: The position of the pit.
    """
```

```
def __init__(self, pos):
    x, y = pos
    self.image = pg.image.load(f"assets/images/{SCALE}x/pit2.png")
    self.pos = Vector2(x, y)
    self.rect = self.image.get_rect(
        center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
    )
    self.mask = pg.mask.from_surface(self.image)
    Collidable_list.append(self)

def update(self, surface):
    surface.blit(self.image, self.rect)
```

<utils/bullet.py>

```
import pygame as pg
from pygame.math import Vector2
from utils import SCALE

Bulletlist = []

"""
This file contains the bullet class.
"""


class Bullet:
    """
    Class for the bullets in the game.

    :param pos: The position of the bullet.
    :param vel: The velocity of the bullet. Vector stores both direction and speed.
    :param type: The type of the bullet.
    """

    def __init__(self, pos, vel, type="red"):
        self.type = type
        self.image = pg.image.load(f"assets/images/{SCALE}x/{type}-bullet.png")
        self.pos = pos * 50 * SCALE + Vector2(25, 25) * SCALE
        self.vel = vel * SCALE
        self.dir = vel.angle_to(Vector2(0, -1))
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.mask = pg.mask.from_surface(self.image)

    def update(self, surface):
        tempimage = pg.transform.rotate(self.image, self.dir)
        self.pos += self.vel
        self.rect.center = self.pos
        surface.blit(tempimage, self.rect)

        if not (0 < self.pos.x < 7 * SCALE * 50 and 0 < self.pos.y < 9 * SCALE *
                50):
            return True
```

<utils.enemy.py>

```
import pygame as pg
from pygame.math import Vector2
from utils import Bulletlist, SCALE

from .bullet import Bullet
from . import player

Enemylist = []

"""
Contains all the enemy classes in the game.
"""


class Enemy:
    """
    Class for the enemies in the game.

    :param type: The type of the enemy.
    :param positions: The positions of the enemy.
    """

    def __init__(
        self,
        type: str,
        positions: list[list[int, int]],
    ):
        self.type = type
        self.positions = positions + positions[::-1]

        self.image = pg.image.load(f"assets/images/{SCALE}x/{self.type}.png")
        self.rect = self.image.get_rect()

        self.counter = 0

        self.mask = pg.mask.from_surface(self.image)

        self.pos = Vector2(self.positions[self.counter])
        self.tgt = Vector2(self.positions[1])

        self.health = 5

        Enemylist.append(self)

    def update(self, surface):
        if self.counter > 120:
```

```
            self.counter = 0
            Bulletlist.append(
                Bullet(self.pos, (player.PlayerPos - self.pos).normalize() * 2)
            )

        self.counter += 1

        if (self.tgt - self.pos).length() < 0.05:
            self.pos = self.tgt
            self.positions.append(self.positions.pop(0))
            self.tgt = Vector2(self.positions[0])
        self.pos = self.pos + (self.tgt - self.pos) * 0.1
        self.rect.center = self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        surface.blit(self.image, self.rect)

        self.collision()
        if self.health <= 0:
            try:
                Enemylist.remove(self)
                return True
            except Exception as e:
                pass

    def collision(self):
        for i in Bulletlist:
            if i.type == "blue":
                if self.mask.overlap(
                    i.mask,
                    (
                        int(i.pos.x - self.pos.x * 50 * SCALE),
                        int(i.pos.y - self.pos.y * 50 * SCALE),
                    ),
                ):
                    Bulletlist.remove(i)
                    self.health -= 1
```

<utils/player.py>

```
import pygame as pg
from pygame.math import Vector2
import utils

SCALE = utils.SCALE

PlayerPos = Vector2(3, 8)

"""
This file contains the player class.
"""


class Player:

    """
    INITALIZE THE PLAYER
    Syntax: Player(x, y, health)

    :param x: x position of the player
    :param y: y position of the player
    :param health: health of the player
    """

    def __init__(self, x=3, y=8, health=5):
        global PlayerPos
        self.image = pg.image.load(f"assets/images/{SCALE}x/player.png")
        self.pos = Vector2(x, y)
        self.rect = self.image.get_rect(
            center=self.pos * 50 * SCALE + Vector2(25, 25) * SCALE
        )
        self.moving = False
        self.vel = Vector2(0, 0)
        self.target = self.pos
        self.targets = []
        self.mask = pg.mask.from_surface(self.image)
        self.health = health
        self.counter = 0
        self.hit = False

    def move(self, direction):
        match direction:
            case "up":
                self.targets.append(Vector2(0, -1))
            case "down":
                self.targets.append(Vector2(0, 1))
```

```
            case "left":
                self.targets.append(Vector2(-1, 0))
            case "right":
                self.targets.append(Vector2(1, 0))

    def is_walkable(self, direction, pos=None):
        if pos == None:
            pos = self.pos

        for i in utils.Collidable_list:
            if i.pos == pos + direction:
                return False
        return True

    def update(self, surface):
        global PlayerPos

        if self.moving:
            if (self.target - self.pos).length() < 0.05:
                self.pos = self.target
                self.moving = False
            self.pos = self.pos + (self.target - self.pos) * 0.3
        else:
            if len(self.targets) != 0:
                direction = self.targets.pop(0)
                target = self.pos + direction
                if (
                    -1 < target.x < 7
                    and -1 < target.y < 9
                    and self.is_walkable(direction)
                ):
                    self.target = target
                    self.moving = True
            self.tgt = self.pos
        PlayerPos = self.pos
        self.rect.center = self.pos * 50 * SCALE + Vector2(25, 25) * SCALE

        self.collision()
        self.shoot_stuff()

        surface.blit(self.image, self.rect)

    def shoot_stuff(self):
        try:
            if self.counter > 10:
                for enemy in utils.Enemylist:
                    if self.pos.x == enemy.tgt.x or self.pos.y == enemy.tgt.y:
                        utils.Bulletlist.append(
                            utils.Bullet(
                                self.pos,
                                (enemy.tgt - self.pos).normalize() * 20,
```

```
                                "blue",
                            )
                        )
                        self.counter = 0
            self.counter += 1
        except:
            pass

    def collision(self):
        for i in utils.Bulletlist:
            if i.type == "red":
                if self.mask.overlap(
                    i.mask,
                    (
                        int(i.pos.x - self.pos.x * 50 * SCALE),
                        int(i.pos.y - self.pos.y * 50 * SCALE),
                    ),
                ):
                    self.hit = True
                    utils.Bulletlist.remove(i)
                    self.health -= 1
```

<utils/txt_button.py>

```
import pygame as pg

"""
This file contains the Ui classes.
"""


class TxtButton:
    """
    INITALIZE THE BUTTON
    Syntax: TxtButton(x, y, txt, color, font)

    :param x: x position of the button
    :param y: y position of the button
    :param txt: text to be displayed on the button
    :param color: color of the text
    :param font: font of the text
    """

    def __init__(self, x, y, txt, color, font):
        self.txt = txt
        self.label = font.render(txt, True, color)
        self.rect = self.label.get_rect(center=(x, y))
        self.bgrect = self.rect.inflate(10, 10)
        self.clicked = False
        self.color = color
        self.font = font

    def update(self, surface, pos):
        action = False

        self.label = self.font.render(self.txt, True, self.color)
        self.bgrect = self.rect.inflate(10, 10)
        self.rect = self.label.get_rect(center=self.rect.center)

        mp = pg.mouse.get_pos()
        if self.rect.collidepoint(mp):
            pg.draw.rect(surface, "#dddddd", self.bgrect)
        if self.rect.collidepoint(pos):
            action = True
        surface.blit(self.label, self.rect)
        return action
```

<Gamedata/highscores.json>

```
{
    "highscore": "0"
}
```

<Gamedata/saves.json>

```
{
    "levelId": 0,
    "score": 0,
    "lives": 5
}
```

<Gamedata/settings.json>

```
{"music": 1}
```

<Gamedata/levels/level0.json>

```
{
  "levelId": 1,
  "playerStartPosition": [3, 8],
  "pitPositions": [
    [0, 5],
    [1, 5],
    [3, 2],
    [3, 5],
    [5, 5],
    [6, 5]
  ],
  "wallPositions": [
    [2, 2],
    [2, 3],
    [2, 4],
    [2, 5],
    [4, 2],
    [4, 3],
    [4, 4],
    [4, 5]
  ],
  "enemies": {
    "enemy1": {
      "type": "glider",
      "positions": [
        [0, 2],
        [0, 3],
        [0, 4]
      ]
    },
    "enemy2": {
      "type": "glider",
      "positions": [
        [1, 4],
        [1, 3],
        [1, 2]
      ]
    },
    "enemy3": {
      "type": "glider",
      "positions": [
        [5, 2],
        [5, 3],
        [5, 4]
      ]
    },
```

```
    "enemy4": {
      "type": "glider",
      "positions": [
        [6, 4],
        [6, 3],
        [6, 2]
      ]
    }
  }
}
```

<Gamedata/levels/level1.json>

```
{
  "levelId": 1,
  "playerStartPosition": [3, 8],

  "pitPositions": [[5, 5]],
  "wallPositions": [[2, 3]],

  "enemies": {
    "enemy1": {
      "type": "glider",
      "positions": [
        [1, 1],
        [2, 1],
        [3, 1],
        [4, 1],
        [5, 1]
      ],
      "speed": 1
    },
    "enemy2": {
      "type": "glider",
      "positions": [
        [1, 4],
        [2, 4],
        [3, 4],
        [4, 4],
        [5, 4]
      ],
      "speed": 1
    }
  }
}
```

<Gamedata/levels/level2.json>

```
{
  "levelId": 2,
  "playerStartPosition": [3, 8],

  "pitPositions": [],
  "wallPositions": [
    [1, 3],
    [5, 3],
    [1, 7],
    [5, 7]
  ],

  "enemies": {
    "enemy1": {
      "type": "glider",
      "positions": [
        [0, 0],
        [1, 0],
        [2, 0],
        [3, 0],
        [4, 0],
        [5, 0],
        [6, 0]
      ]
    },
    "enemy2": {
      "type": "glider",
      "positions": [
        [6, 0],
        [5, 0],
        [4, 0],
        [3, 0],
        [2, 0],
        [1, 0],
        [0, 0]
      ]
    },
    "enemy3": {
      "type": "glider",
      "positions": [
        [6, 8],
        [5, 8],
        [4, 8],
        [3, 8],
        [2, 8],
        [1, 8],
```

```
        [0, 8]
      ]
    },
    "enemy4": {
      "type": "glider",
      "positions": [
        [0, 8],
        [1, 8],
        [2, 8],
        [3, 8],
        [4, 8],
        [5, 8],
        [6, 8]
      ]
    },
    "enemy5": {
      "type": "glider",
      "positions": [
        [0, 4],
        [1, 4],
        [2, 4],
        [3, 4],
        [4, 4],
        [5, 4],
        [6, 4]
      ]
    },
    "enemy6": {
      "type": "glider",
      "positions": [
        [6, 4],
        [5, 4],
        [4, 4],
        [3, 4],
        [2, 4],
        [1, 4],
        [0, 4]
      ]
    }
  }
}
```

<Gamedata/levels/level3.json>

```
{
  "levelId": 1,
  "playerStartPosition": [3, 7],
  "pitPositions": [
    [0, 5],
    [1, 2],
    [1, 3],
    [1, 4],
    [1, 5],
    [5, 2],
    [5, 3],
    [5, 4],
    [5, 5],
    [6, 5]
  ],
  "wallPositions": [[3, 4]],
  "enemies": {
    "enemy1": {
      "type": "glider",
      "positions": [
        [0, 1],
        [0, 2]
      ]
    },
    "enemy2": {
      "type": "glider",
      "positions": [
        [0, 3],
        [0, 4]
      ]
    },
    "enemy3": {
      "type": "glider",
      "positions": [
        [3, 3],
        [3, 2],
        [3, 1]
      ]
    },
    "enemy4": {
      "type": "glider",
      "positions": [
        [6, 1],
        [6, 2]
      ]
    },
```

```
    "enemy5": {
      "type": "glider",
      "positions": [
        [6, 3],
        [6, 4]
      ]
    }
  }
}
```

<Gamedata/levels/level4.json>

```
{
  "levelId": 1,
  "playerStartPosition": [3, 7],
  "pitPositions": [
    [1, 2],
    [2, 6],
    [2, 7],
    [3, 6],
    [4, 6],
    [4, 7],
    [5, 2]
  ],
  "wallPositions": [
    [0, 4],
    [1, 3],
    [1, 4],
    [3, 2],
    [5, 3],
    [5, 4],
    [6, 4]
  ],
  "enemies": {
    "enemy1": {
      "type": "glider",
      "positions": [
        [0, 1],
        [0, 2],
        [0, 3]
      ]
    },
    "enemy2": {
      "type": "glider",
      "positions": [
        [0, 8],
        [1, 8],
        [2, 8]
      ]
    },
    "enemy3": {
      "type": "glider",
      "positions": [
        [2, 3],
        [2, 4]
      ]
    },
    "enemy4": {
      "type": "glider",
```

```
      "positions": [
        [3, 3],
        [3, 4]
      ]
    },
    "enemy5": {
      "type": "glider",
      "positions": [
        [4, 3],
        [4, 4]
      ]
    },
    "enemy6": {
      "type": "glider",
      "positions": [
        [6, 1],
        [6, 2],
        [6, 3]
      ]
    },
    "enemy7": {
      "type": "glider",
      "positions": [
        [6, 8],
        [5, 8],
        [4, 8]
      ]
    }
  }
}
```

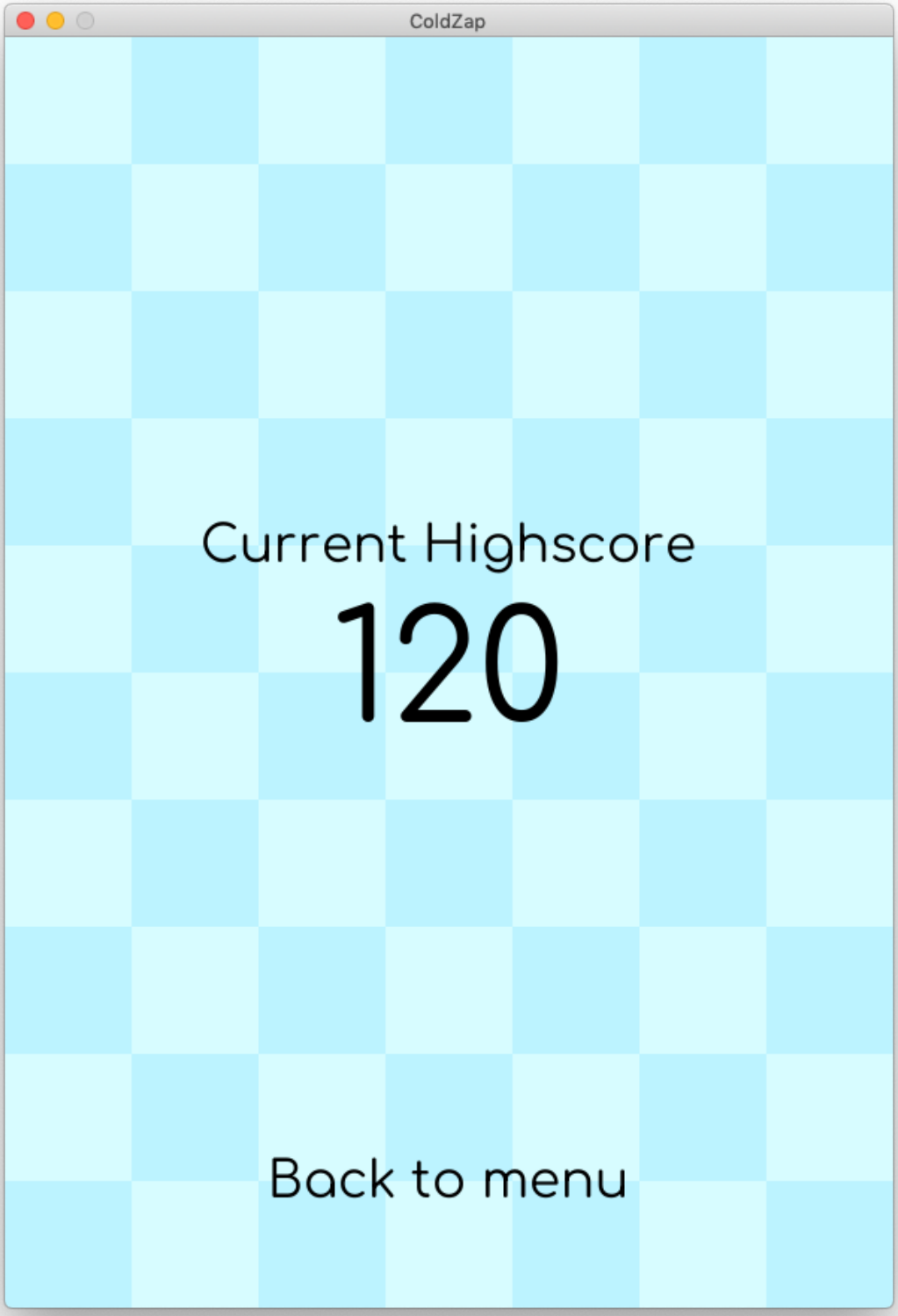
<Output>

The following images detail the various scenes obtained during the gameplay of ColdZap.

img-1	–	main menu
img-2	–	high scores
img-3	–	settings menu
img-4	–	Lv-1
img-5	–	Lv-2
img-6	–	Lv-3
img-7	–	Lv-4

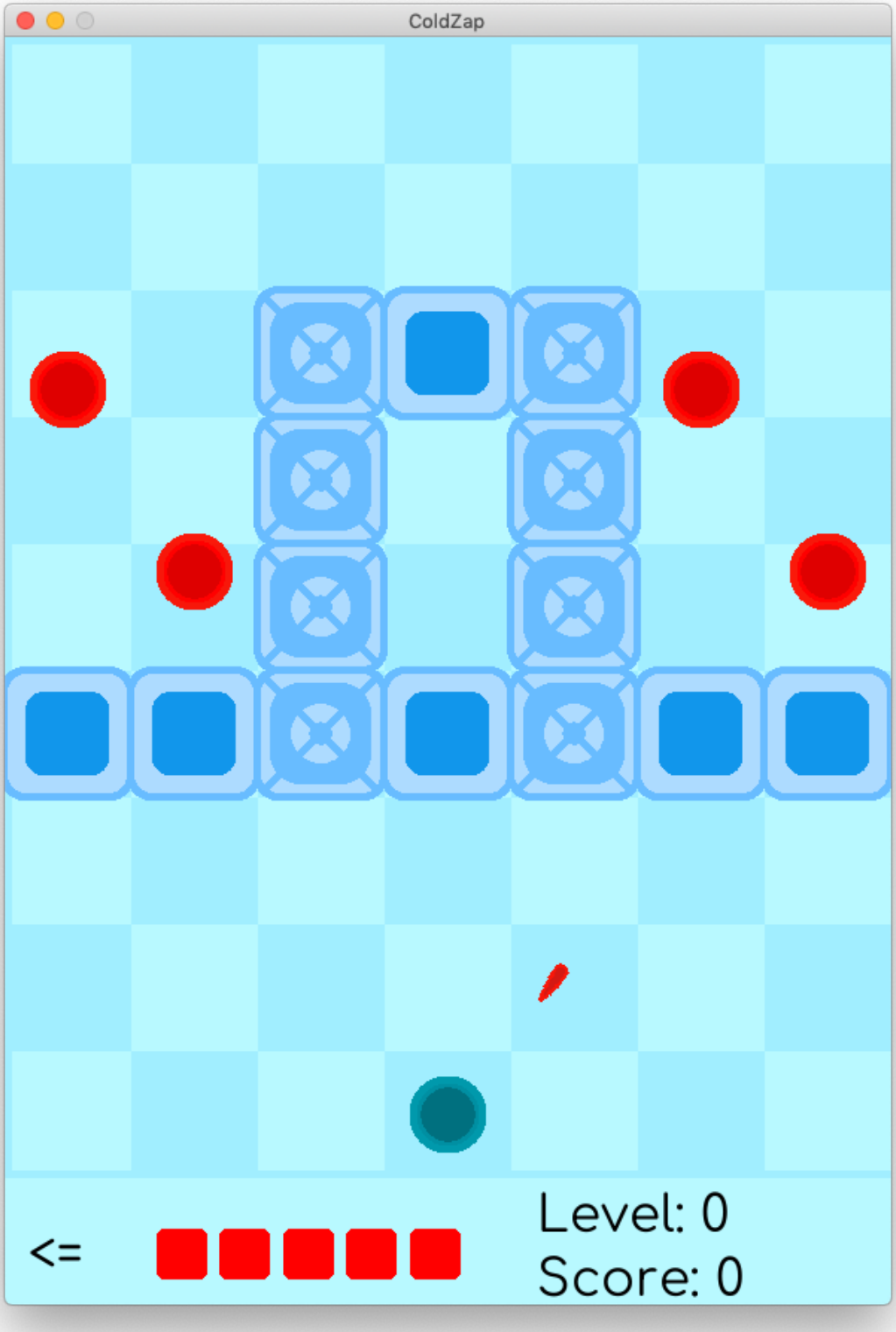

ColdZap
ColdZap
New Game
Load Game
Highscores
Settings
Quit

Img-1

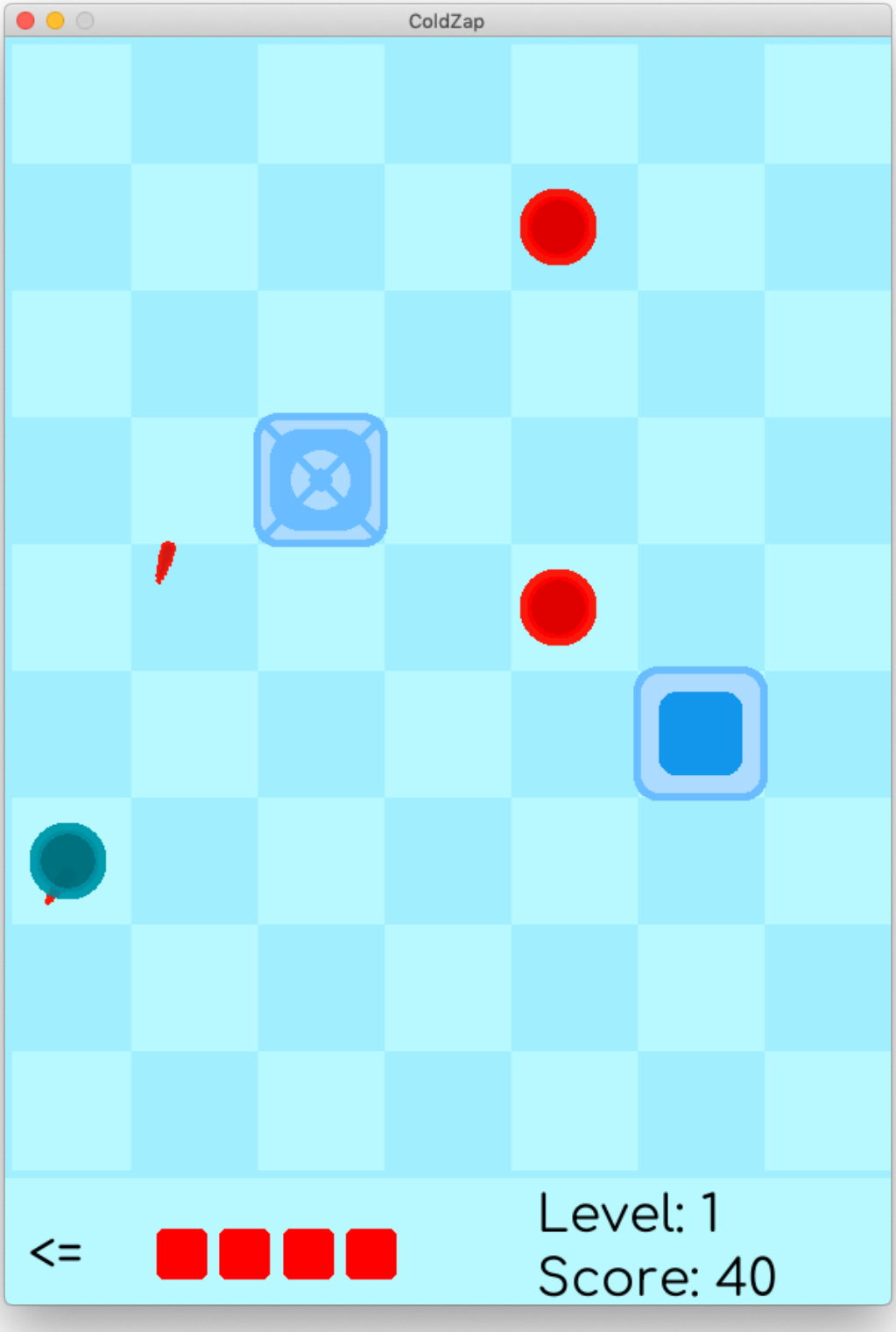


Img-2

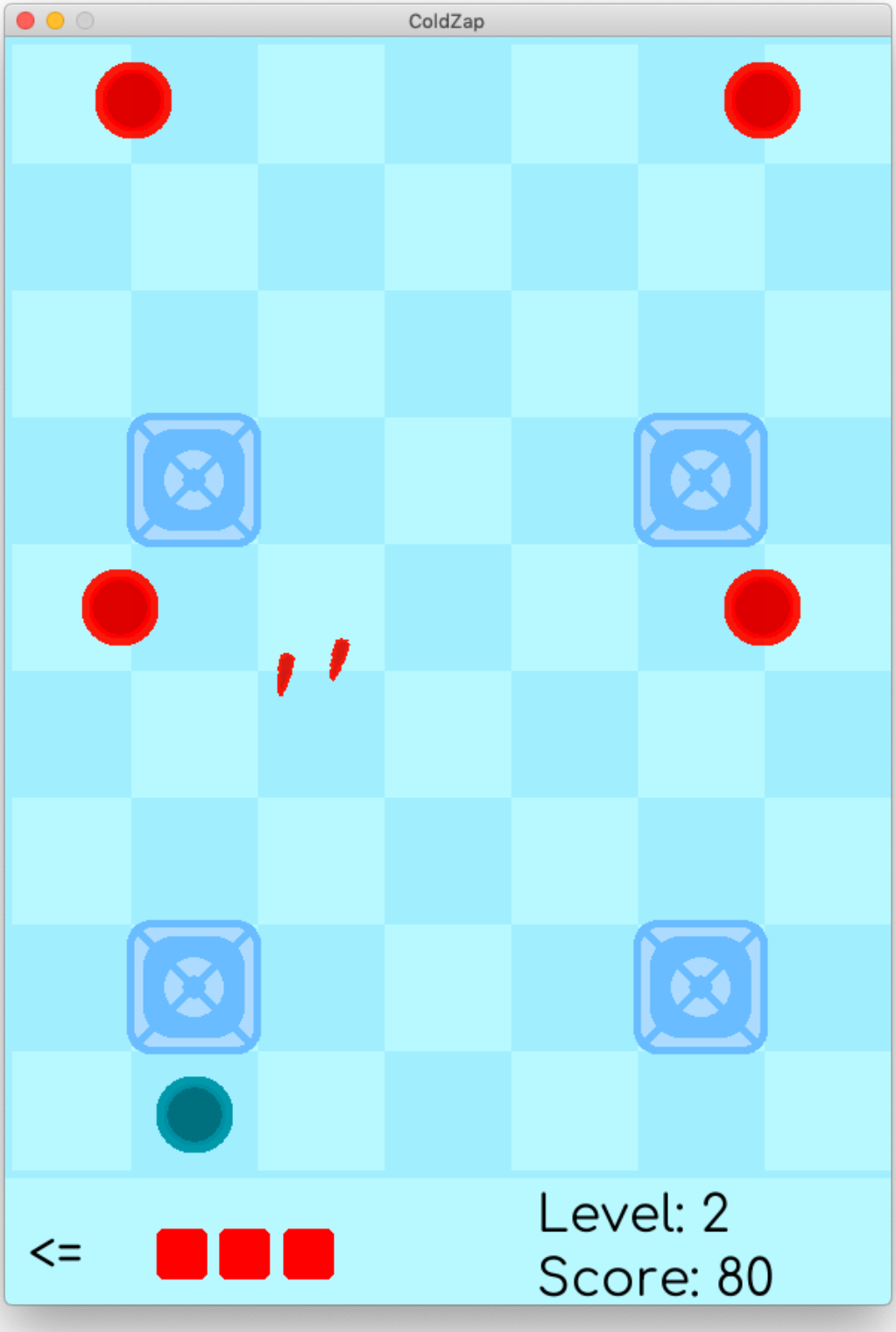
Img-3



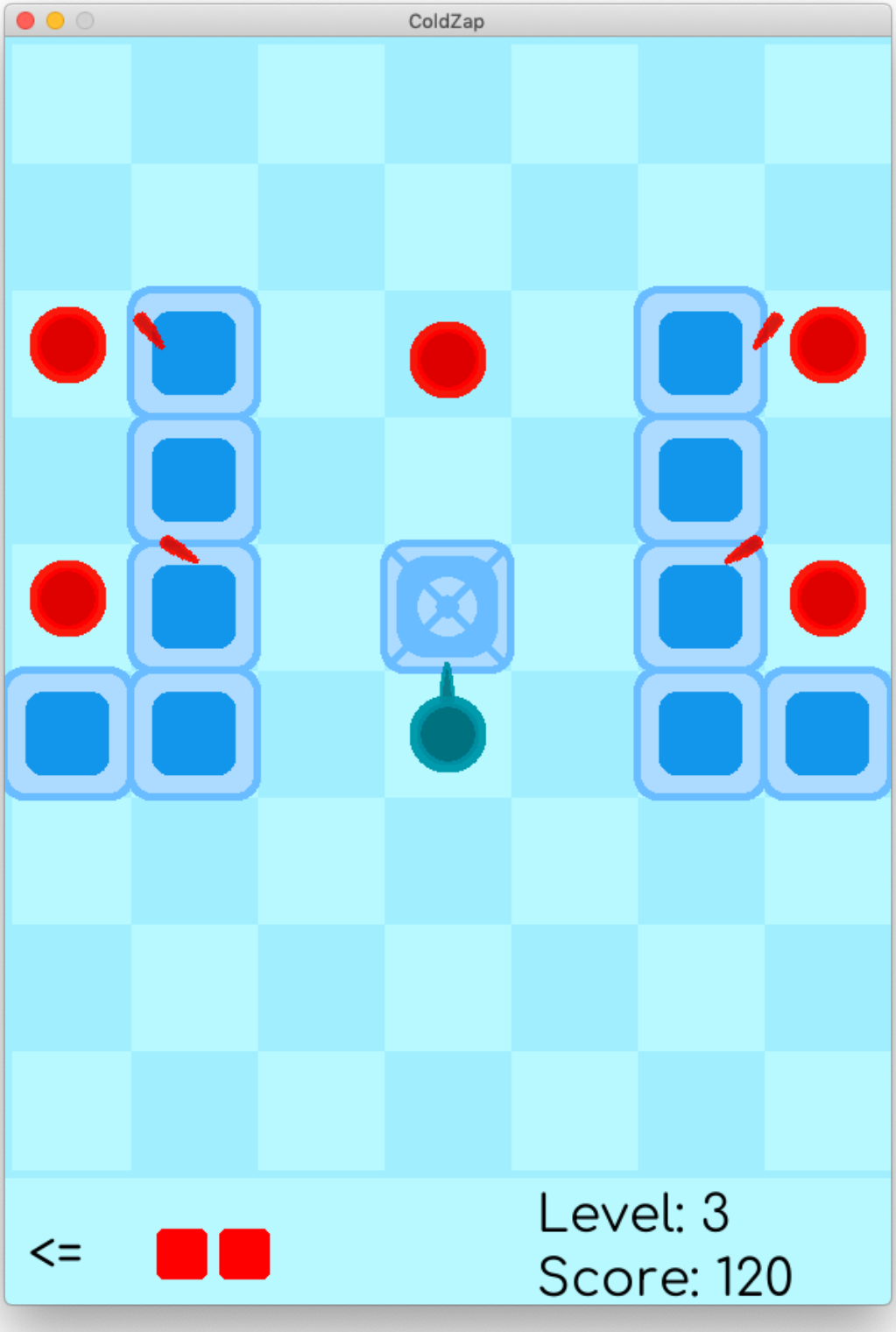
Img-4



Img-5



Img-6



Img-7

<References>

Python – docs.python.org
Pygame – pygame.org/docs/