GAME DESIGN DOCUMENT
INCEPTION TO TTFE

Abdigaliyev A. Zhasulan,
Kashkenov S. Madiyar,
Kenbay B. Ilyas,
Smagulov A. Danial
(SE-2304)

I. VISION, SCOPE, AND GOAL OF PROJECT

1. VISION.

The game Inception to TTFE is designed as an addictive and easy to master puzzle game that offers players a fun way to improve their logical thinking and strategic planning skills. Games like 2048 are popular for their simplicity and depth, offering endless opportunities to improve personal records. The project aims to create a fully functional 2048 game on the Pygame platform, providing smooth and responsive game mechanics.

2. SCOPE.

The project includes the development of basic game mechanics such as shifting and merging tiles, generating new tiles after each turn, and scorekeeping. The game will include a user interface that displays the current score and provides the option to start the game over. What sets this game apart from other 2048 games is that it has a timer after which the game ends.

3. GOAL OF PROJECT.

- Complete prototype of the game.

II. TEAM ROLES/PARTICIPATIONS/CONTRIBUTIONS.

Team members	Role
Abdigaliyev A. Zhasulan	2D Designer, Sound Designer, Programmer, GDD
Kashkenov S. Madiyar	Programmer
Kenbay B. Ilyas	Programmer
Smagulov A. Danial	Programmer

III. KEY REQUIREMENTS SUMMARY.

1. DESIGN.

To design our game 2048, we will analyse the existing version of the game (https://2048game.com). Based on the data collected, we will develop a prototype.

During the development process, we will conduct regular playtesting to ensure that the game works properly and fits our plans.

2. TECHNOLOGY.

To create our game, we use the Pygame library, which provides tools for developing 2D games in Python.

3. PEOPLE

Each team member performs a specific role in the project. The team includes Abdigaliev Zhasulan, who is a 2D designer, sound designer and author of cutscenes, Kenbai Ilyas, a logic and interface programmer, Danial Smagulov, a programmer who created the board and control, Madiyar Kashkenov, a programmer of the database and the main block of the game. We all took part in the programming, and we all made an equal contribution to the creation of the game.

4. TIMEFRAME.

It took one day to analyze the existing game and create a concept. It took two weeks to study tutorials and develop the game.

IV. CONCEPT SUMMARY.

1. TITLE.

The game is called "Inception to TTFE", which reflects the main plot of the game - to infiltrate the TTFE organization to destroy it from the inside. It is not clear to a simple player what TTFE means, but we, as developers, can whisper that TTFE is an abbreviation of 2048 (Two thousand forty-eight).

2. GAMEPLAY HOOK.

2048 is a unique numerical puzzle game that forces players to think strategically, anticipate movements and decide how best to combine tiles to achieve a high score. The main hook of the game is a mixture of ease of learning and difficulty in mastery, as the player's time is limited.

3. GENRE

The genre of this game is puzzle-solving.

4. THEME.

The main theme of the game is abstract thinking and a space of possibilities, where every move can both bring the player closer to victory and increase the complexity of the game. The game inspires strategic planning and foresight.

5. PLATFORM.

The game is available to play only on PC.

V. GAME STORY.

The main plot of the game - to infiltrate the TTFE organization to destroy it from the inside. The player must be interviewed through a test called "2048". There is

no joke in such a mission, so the atmosphere of the game is dark, and the music is melancholic. The player's goal is to score 2048 in 4 minutes.

VI. GAMEPLAY SYSTEM.

1. RULES AND MECHANICS.

Basic rules of the game: The player must connect tiles with the same numbers, which leads to a doubling of the number on the tile. The goal of the game is to create a tile with the number 2048. The player can move the tiles in four directions — up, down, left and right. Each turn leads to the appearance of a new tile with the number 2 or 4 in a random empty cell. The game ends when there are no empty cells on the field, and it is impossible to make moves that lead to the merging of tiles. The rule of the game is written in the first cutscene.

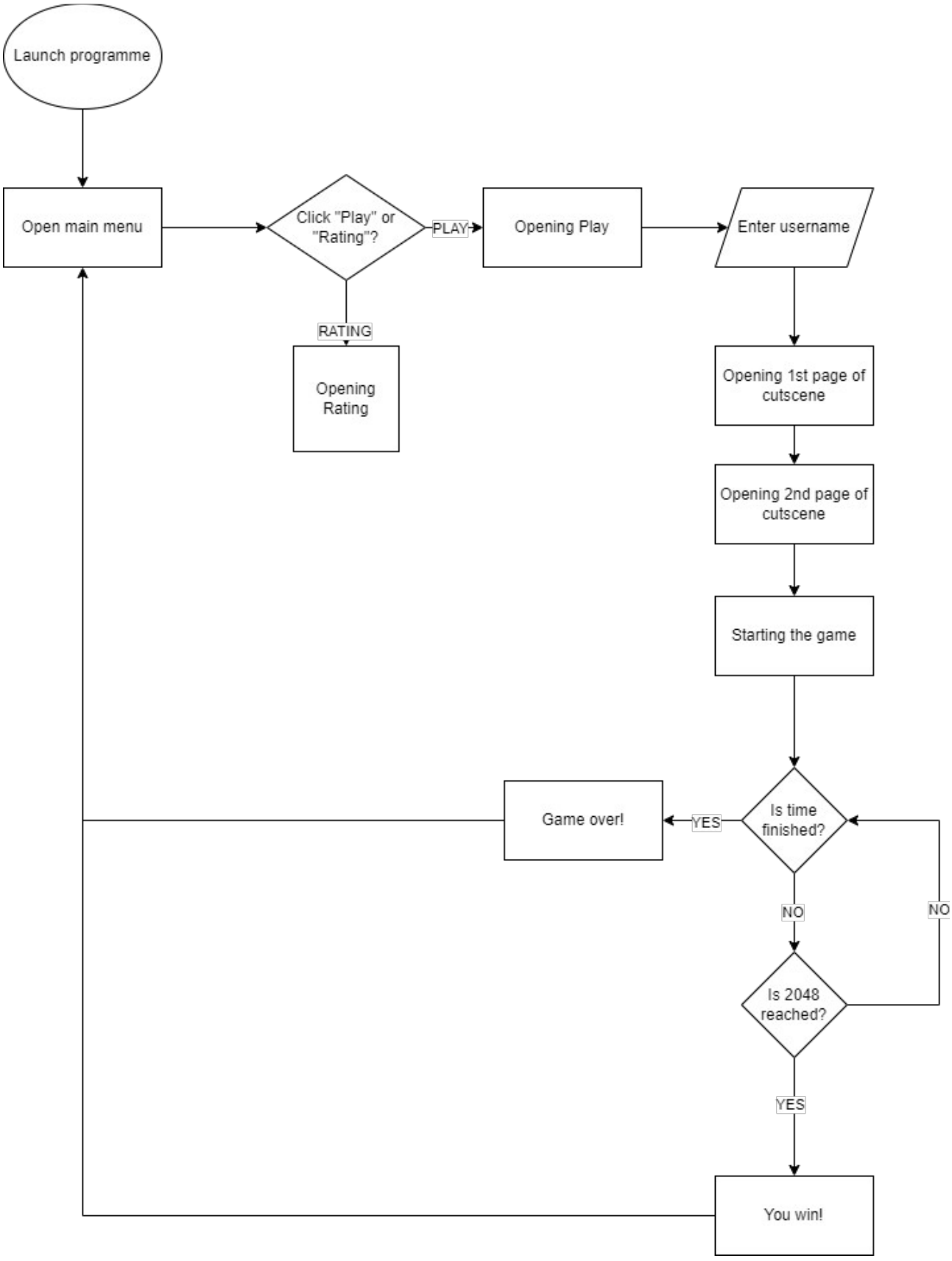
2. GAMEPLAY MODE.

The game will have one game mode. This mode will offer a high level of difficulty due to the time limit because it is difficult to get into the organization. This allows players to find strategies to achieve the goal in the shortest possible time.

3. REWARD SYSTEM.

When a player successfully creates a tile with a higher number, he gets points that are added to his total score. Reaching the 2048 tile is the main and only goal. For players who achieve high results, there is a leaderboard system where they can compare their achievements with the results of other players locally.

4. LOGIC OF GAME.



5. USING BASIC DATA STRUCTURES.

Basic data structures such as lists, tuples, dictionaries and sets were used to manage the game state. There are many cases where we used basic data structures. Below are screenshots of some of them.

```
# A dictionary of color codes used in the game, with keys as numbers for number blocks or specific rows.
COLORS = {
    0: "#2e2e2e",
    2: "#3d2626",
    4: "#4e2020",
    8: "#600000",
    16: "#7a1a1a",
    32: "#8b0000",
    64: "#9f1e20",
    128: "#a22828",
    256: "#ab3232",
    512: "#b33d3d",
    1024: "#bf4747",
    2048: "#cb5151",
    "WHITE": "#ebeeff",  # Color for text.
    "GRAY": "#aebad0",  # Color for text.
    "BLACK": "#000000"
```

```
# If the board is provided, use it; otherwise, create an empty one
if mas is not None:
    self.__mas = mas
else:
    self.__mas = [
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
    ]
```

```python
# Returns a tuple with two values to substitute the text of the values on the screen.
2 usages  Ilyas
def get_size_font(score_size: int, score_top_size: int) -> tuple[int, int]:
    size_score = len(str(abs(score_size)))
    score_top_size = len(str(abs(score_top_size)))

    if size_score == 6 and score_top_size == 6:
        return 20, 20
    if size_score == 6 or score_top_size == 6:
        if size_score == 6 and score_top_size != 6:
            return 22, 25
        return 25, 20
    return 25, 25
```

6. USING CONDITIONAL STATEMENTS.

There are many cases where we used conditional statements. Below are screenshots of some of them.

```python
insert_result(self.username, self.score)
make_decision = False
while not make_decision:
    for event in pg.event.get():
        if event.type == pg.QUIT:  # Clicked on the cross
            self.username = None
            pg.quit()
            sys.exit()
        elif event.type == pg.KEYDOWN:
            if event.key == pg.K_ESCAPE:
                self.username = None
                pg.quit()
                sys.exit()
            elif event.key == pg.K_RETURN:
                self.username = None
                make_decision = True
            elif event.key == pg.K_BACKSPACE:
                make_decision = True
        elif event.type == pg.MOUSEBUTTONDOWN and repeat_box.collidepoint(event.pos):
            make_decision = True
```

```
elif event.type == pg.KEYDOWN:
    if event.key == pg.K_ESCAPE:
        self.save_game()
        pg.quit()
        sys.exit()
    elif event.key in (pg.K_LEFT, pg.K_a):  # Left
        self.copy_board = quick_copy(self.board)
        self.board.move_left(self)
    elif event.key in (pg.K_RIGHT, pg.K_d):  # Right
        self.copy_board = quick_copy(self.board)
        self.board.move_right(self)
    elif event.key in (pg.K_UP, pg.K_w):  # Up
        self.copy_board = quick_copy(self.board)
        self.board.move_up(self)
    elif event.key in (pg.K_DOWN, pg.K_s):  # Down
        self.copy_board = quick_copy(self.board)
        self.board.move_down(self)
    self.update()
    if self.is_victory():
        self.draw_victory()
```

7. USING LOOPS.

There are many cases where we used loops. Below are screenshots of some of them.

```
# Main game loop
1 +2
def run(self) -> None:
    try:
        while True:
            pg.mixer.music.stop()
            self.load_game()
            if self.username is None:
                play_music("menu")
                self.draw_menu()
                self.show_cutscene_one()
                self.show_cutscene_Two()
                play_music("game")
            self.draw_main()
            while self.board.are_there_zeros() and self.board.can_move() and self.time_check():
                pg.display.update()
                if self.handle_events() is True:
                    pg.mixer.music.stop()
                    break
                self.update_timer()
                self.draw_timer()
                pg.display.update()
                self.clock.tick(self.framerate)
            else:
                self.draw_game_over()
    except Exception as exc:
        self.save_game()
        raise exc from None
```

```
# Wait for the "Start" button to be pressed
waiting_for_input = True
while waiting_for_input:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()
            sys.exit()
        if event.type == pg.MOUSEBUTTONDOWN:
            if start_btn_rect.collidepoint(pg.mouse.get_pos()):
                waiting_for_input = False
```

8. USING FUNCTIONS.

There are many cases where we used functions. Below are screenshots of some of them.

```
# Abstract method for updating the game status
 MeaningfulName
@abstractmethod
def update(self) -> None:
    """Updating the game status."""
# Abstract method for handling player input events
 MeaningfulName
@abstractmethod
def handle_events(self) -> bool:
    """Handles actions entered by the player."""
# Abstract method for running game
 MeaningfulName
@abstractmethod
def run(self) -> None:
    """Launches the game."""
```

```
# Method to load the last saved game
1 usage  1 *
def load_game(self) -> None:
    path = Path.cwd()

    if "save.txt" in os.listdir(path):
        with open("save.txt") as file:
            data = json.load(file)
            self.board = GameBoard(data["board"])
            self.score = data["score"]
            self.username = data["user"]
        full_path = path / Path("save.txt")
        Path(full_path).unlink()
    else:
        super().__init__()
        self.board = GameBoard()
        self.move_mouse = False
```

9. ERROR HANDLING.

There are many cases where we used functions. Below are screenshots of some of them.

```
# Method returns the result of the top 3 players.
5 usages  1
def get_best(count: int = 0) -> dict:
    try:
        assert -1 <= count <= 3
    except AssertionError as exc:
        msg = "Invalid argument count, must be no more than 3 and no less than -1"
        raise ValueError(msg) from exc
```

```
def run(self) -> None:
    try:
        while True:
            pg.mixer.music.st
            self.load_game()
            if self.username
                play_music("m
                self.draw_men
                self.show_cut
                self.show_cut
                play_music("g
            self.draw_main()
            while self.board.
                pg.display.up
                if self.handl
                    pg.mixer.
                    break
                self.update_t
                self.draw_tim
                pg.display.up
                self.clock.ti
            else:
                self.draw_gam
    except Exception as exc:
        self.save_game()
        raise exc from None
```

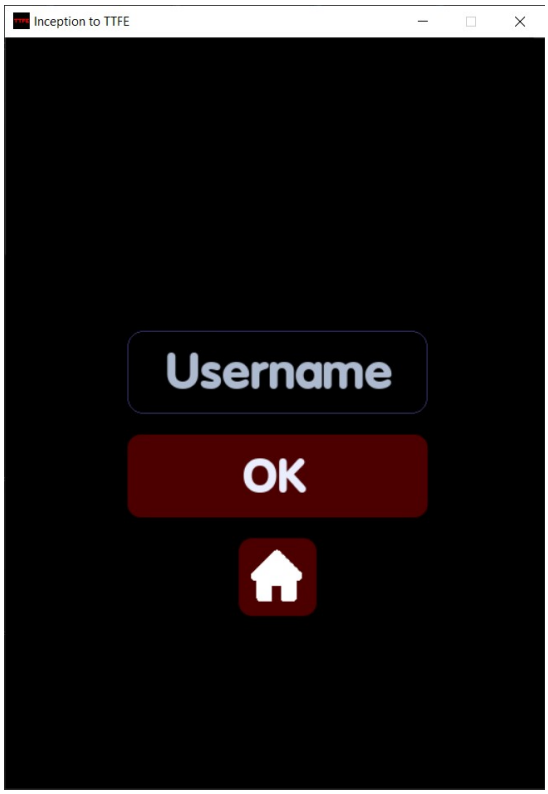
VII. ART DESIGN (VISUAL AND AUDIO).

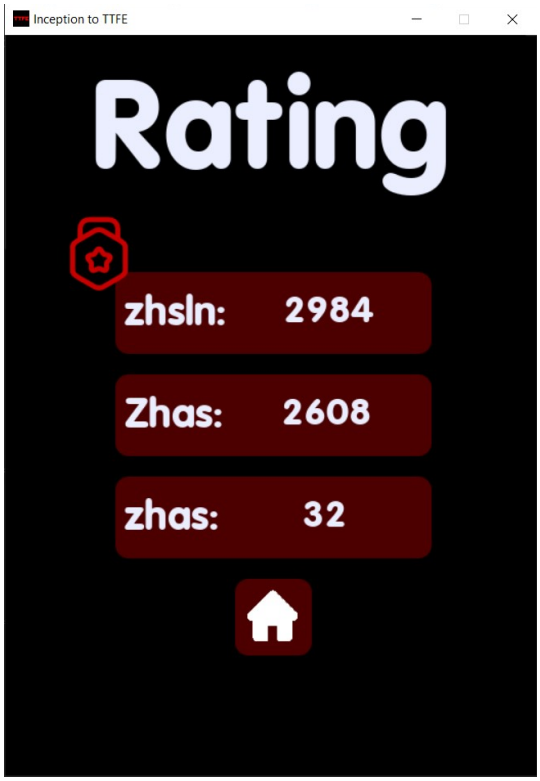
1. VISUAL DIRECTION.

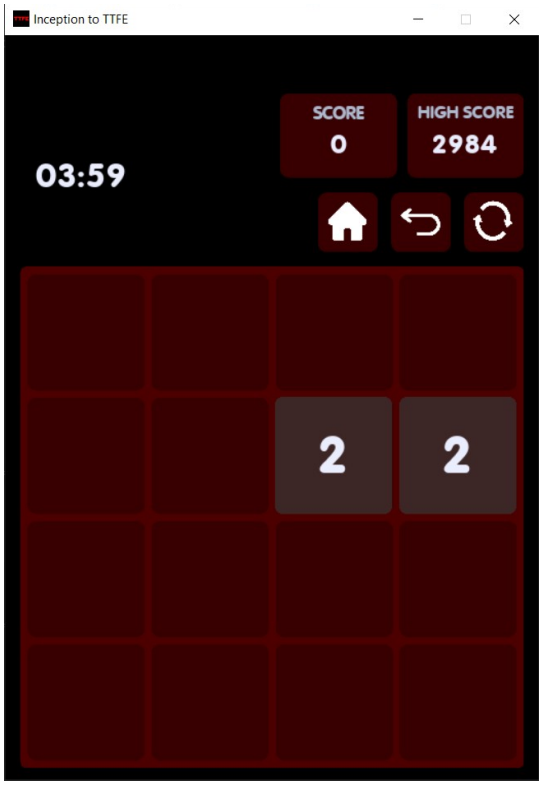
The visual of the game was created using Figma. Visually, the game looks gloomy to match the plot.

2. USER INTERFACE (UI).

User interface is intuitive and visually appealing (The size of the screenshot is reduced and does not correspond to the real size of the game window).




Inception to TTFE
Rating
zhsln: 2984
Zhas: 2608
zhas: 32


Inception to TTFE
SCORE
0
HIGH SCORE
2984
03:59
2
2

3. GRAPHICS.

The game's graphics are drawn using functions. Below are screenshots of some of these functions.

```python
# draws timer in board
1 usage  llyas
def draw_timer(self) -> None:
    pg.draw.rect(self.screen, config.COLORS["BLACK"], rect: (30, 110, 220, 50))  # Adjust dimensions as needed

    minutes = self.timer // 60
    seconds = self.timer % 60
    timer_text = f"{minutes:02d}:{seconds:02d}"

    timer_surface = pg.font.Font(self.generalFont, size: 32).render(timer_text, antialias: True, config.COLORS["WHITE"])
    self.screen.blit(timer_surface, dest: (30, 100))  # Adjust the position as needed
```

```python
# draws game boards menu
3 usages  llyas
def draw_main(self) -> None:
    self.screen.blit(
        pg.transform.scale(pg.image.load(resource_path(BG_PATH / Path("BG.jpg"))), size: (self.width, self.height + 2)),
        dest: (0, 0),
    )
    self.screen.blit(
        pg.transform.scale(pg.image.load(resource_path(ELEMENTS_PATH / Path("around_arrow.png"))), size: (43, 43)),
        dest: (453, 159),
    )
    self.screen.blit(
        pg.transform.scale(pg.image.load(resource_path(ELEMENTS_PATH / Path("arrow.png"))), size: (58, 58)),
        dest: (374, 154),
    )
    self.screen.blit(
        pg.transform.scale(pg.image.load(resource_path(ELEMENTS_PATH / Path("home.png"))), size: (38, 38)),
        dest: (314, 162),
    )

    self.screen.blit(
        pg.font.Font(self.generalFont, size: 17).render( text: "HIGH SCORE", antialias: True, config.COLORS["GRAY"]),
        dest: (402, 55),
    )
    self.screen.blit(
        pg.font.Font(self.generalFont, size: 18).render( text: "SCORE", antialias: True, config.COLORS["GRAY"]),
        dest: (300, 55),
    )

    best_score = get_best(1)["score"]
    high_score = 0 if best_score == -1 else best_score
```

4. AUDIO DIRECTION.

Background music and game music have been integrated to enhance the gaming experience. To match the plots, the works of Sergei Vasilyevich Rachmaninoff Prelude in G Minor No. 5 op. 23 and the Island of the Dead op. 29 were used.

```
# Dictionary containing paths to audio tracks for different game states
audio_tracks = {
    "game": "music/Rachmaninoff - Prelude in G minor, op. 23, No. 5.mp3",
    "menu": "music/С. В. Рахманинов - Остров мёртвых.mp3",
}

# Function to play music based on the provided track ID
2 usages  1
def play_music(track_id):
    if track_id in audio_tracks:
        pg.mixer.music.load(audio_tracks[track_id])
        pg.mixer.music.play(-1)
    else:
        print(f"Music with ID '{track_id}' not found.")
```

VIII. APPENDECIES.

To create this game, a Youtube video from "The Italian Makers" (https://www.youtube.com/watch?v=qwJ9w5bmKZU) was viewed.

The source code of our game is available to everyone and published on GitHub: https://github.com/manInTheJacket/2048