Dyber

פרויקט גמר מגמת הנדסת תוכנה
תיכון בגין ראש העין

גיא אביטל - 213120710
מנחה: לינה שמידט
הגשה: יוני 2021

המוצר

המוצר אותו כתבתי הוא משחק מהיר קצב הנקרא Dyber. משחק זה נועד ל-4 שחקנים. המוצר מחבר בין 4 מחשבים לשרת ראשי המכיל את כל המידע על החדרים. כל חיבור לשרת מועבר לרשימת שחקנים של חדר פעיל בו נשמר כל המידע על המשחק והוא מתפעל אותו. לשם כך כתבתי חלק קוד המחבר את השחקנים לשרת, וקטע קוד המחבר שחקנים לחדר המשחק הפעיל.

Python

בחרתי לכתוב את הפרויקט ב - Python. בחרתי בשפה זו כיוון והיא נוחה לשימוש ונלמדה אקטיבית בבית הספר כך שאליה התרגלתי. בנוסף היא בעלת ספריות רבות המקלות על כתיבת תוכנות מסובכות וגדולות. לדוגמאת Pygame, ספריה לבניית משחקים בה השתמשתי בכדי ליצור את הגרפיקה של המשחק.

Pygame

ספרית הממשק הגרפי שבה השתמשתי כדי ליצור את המשחק בצורה נוחה ודרכה עיצבתי אותו כך שירוץ חלק בלי קשר למחשב.

Socket

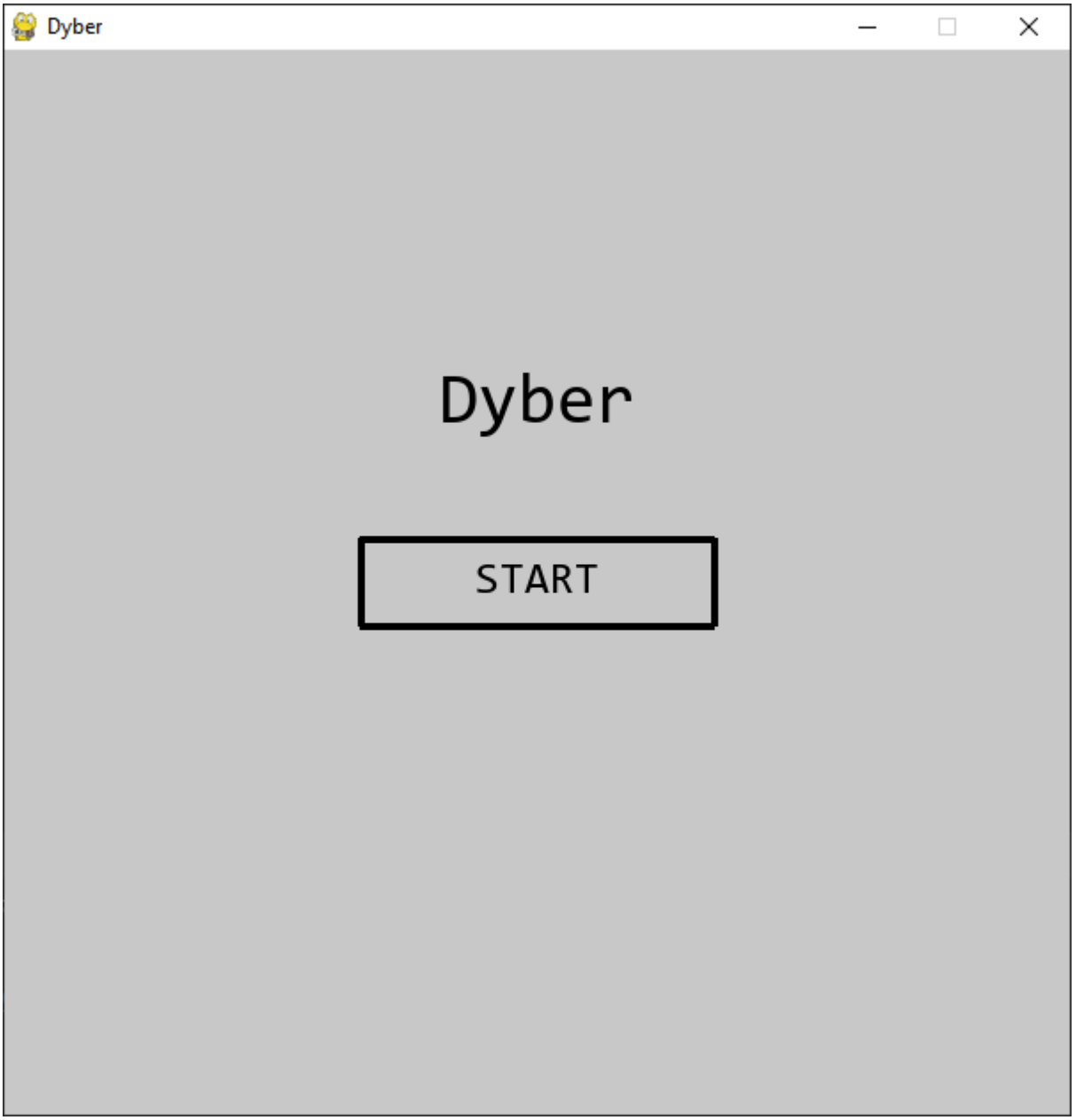
ספריה בשפה ממנה ניתן להעביר מידע בין מחשבים דרך הרשת. השתמשתי בספריה זו לחיבור ותקשורת בין מחשבי השחקנים.

Threading

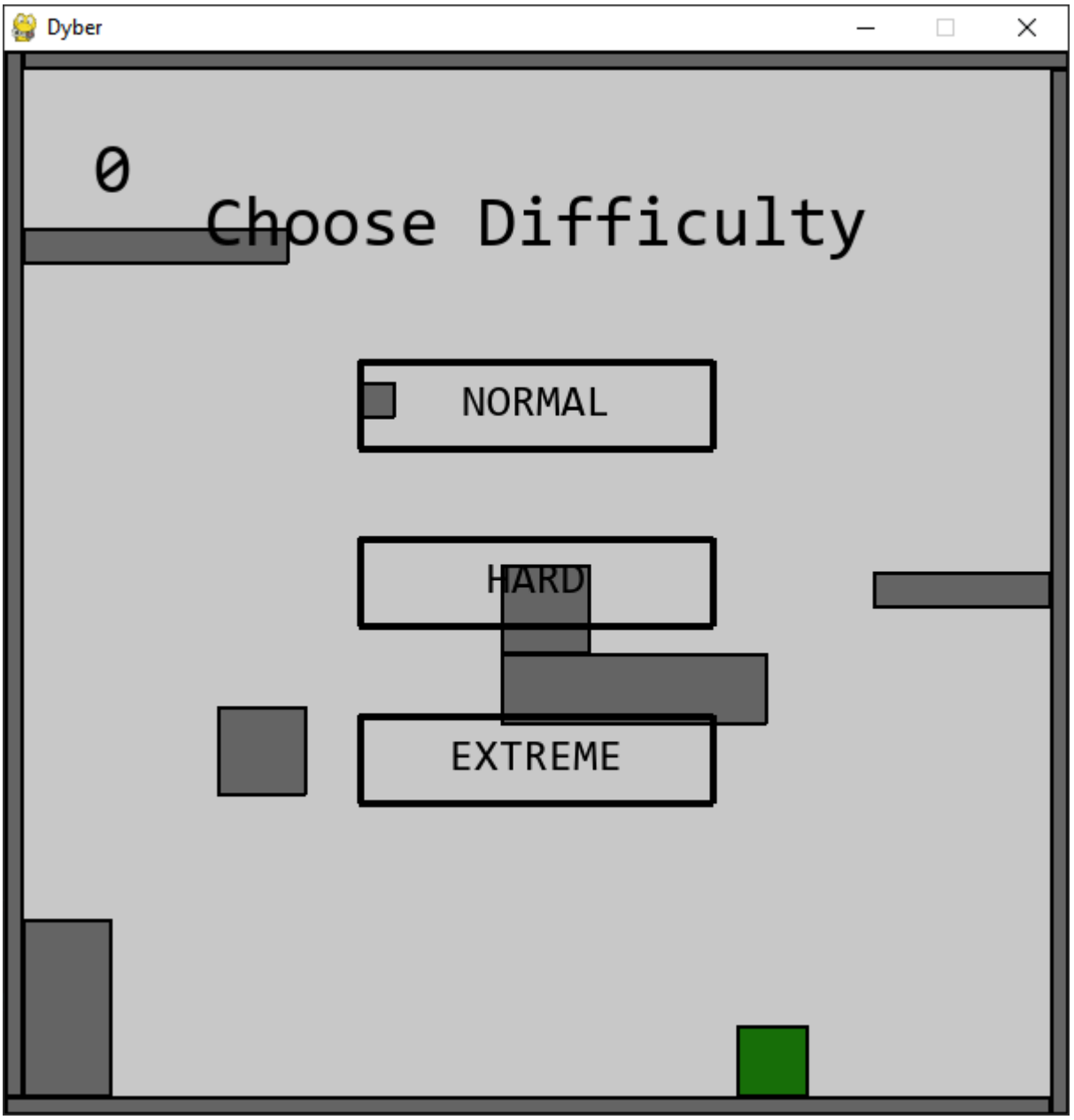
ספריה המאפשרת הרצה של מספר קטעי קוד בו זמנית. השתמשתי בה על מנת ליצור חדרים רבים הרצים זה לצד זה וכדי ליצור תקשורת רציפה ושוטפת ללא עצירות מיותרות של המשחק.

עיצוב

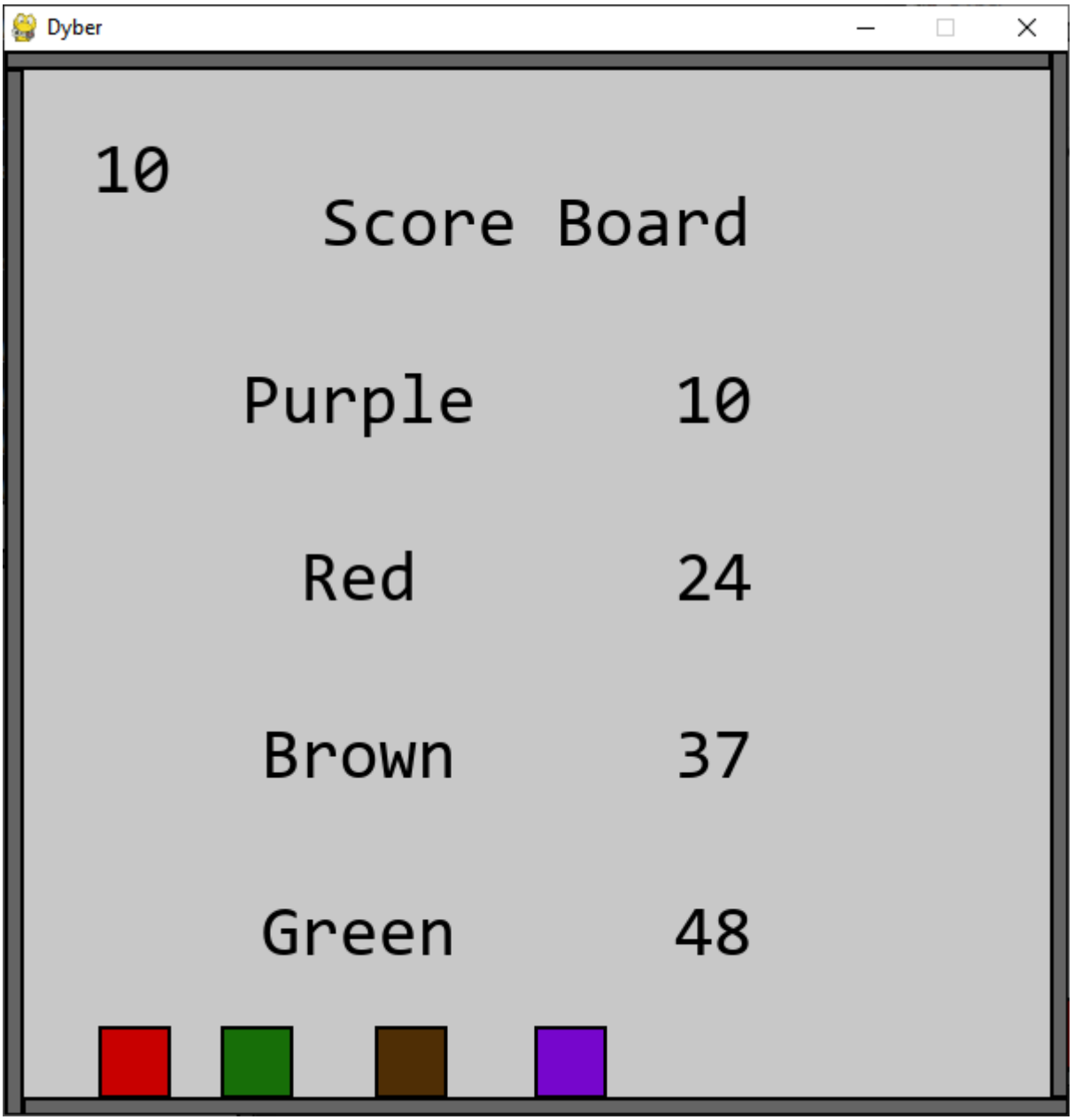
בתחילת המשחק שחקן נכנס למסך הנקרא Welcome Screen ומטרתו לברך את פני השחקן לפני תחילת המשחק. יוצג לשחקן כפתור המאפשר לו להתחיל את המשחק. כפתור זה מחבר את השחקן לחדר פנוי אצל השרת ומכניס אותו לחלון המשחק.



היות והשחקן הינו השחקן הראשון שהתחבר לחדר המשחק, הוא יתבקש לבחור דרגת קושי לכל החדר, בתחילת המשחק הוא בעצמו יקבע את מספר השלבים אותם עוברים שחקני החדר.



בסוף השלבים, השחקנים יגיעו לחדר בו מוצגות התוצאות. בחדר ישנו לוח המציג את הזמן שלקח לכל שחקן לעבור את כל השלבים בדרגת הקושי שצוינה לפי סדר יורד.



דיון

בעיה שנתקלתי בה במהלך כתיבת הקוד הייתה חוסר הסנכרון בין השחקנים. כיוון שהשחקנים תמיד מצוירים על המסך (עם נקודות ציון מסוימות) ולא בהכרח נמצאים באותו שלב כמו שאר השחקנים, יצא מצב בו כל השחקנים צוירו באותו חלון בעוד הרקע הינו השלב של השחקן המשחק באותו מחשב בעוד שהשלב בו שאר השחקנים נמצאים יכול להיות שונה לגמרי. פתרתי בעיה זו בעזרת שילוב של המשחק בקובץ הלקוח עם החדר בקובץ השרת, כלומר, החלטתי לשמור את מספר השלב בו נמצא כל שחקן בחדר המשחק, כך שכשנקודות הציון שלו נשלחות לשאר השחקנים הם מקבלים גם את השלב בו הוא נמצא וכך יודעים האם לצייר אותו איתם או לא. מה שפעולה זו מאפשר לנו לראות במשחק היא תחושה של רצף בין השלבים כששחקן אחד מתקדם יותר מהאחרים ו-"משאיר אותם מאחור" כמו שהיה קורה במצב פיזי בעולם מציאותי.

הסבר תקשורת וציור

```python
class Renderer(Thread):
    """
    A thread to run the render process to increase frame rate whilst keeping 60 FPS update rate
    """
    def run(self) -> None:
        """
        Render all game objects, players and menus
        """
        while Game.is_running:
            Game.surface.fill(Game.COLOR_BACKGROUND)

            # when in welcome screen
            if Game.welcome:
                Game.draw_text(Game.TITLE, (Game.WIDTH / 2, Game.HEIGHT / 2 - 100), Game.header_font)
                Game.start_btn.render()
            else:  # when in game screen
                for obj in Game.get_level_objects():
                    obj.render(Game.surface, pyg)

                for opponent in Game.opponents.values():
                    if opponent != Game.player1 and opponent.level == Game.current_level:
                        opponent.render(Game.surface, pyg)

                Game.player1.render(Game.surface, pyg)

                # show time since game start
                past = Game.header_font.render(str(Game.frame_cycles), True, Game.COLOR_BLACK)
                Game.surface.blit(past, (50, 50))

                # if in god test mode -> show indication
                if Game.god_mode:
                    god = Game.header_font.render('GOD', True, Game.COLOR_BLACK)
```

```
                Game.surface.blit(god, (500, 50))

            # in difficulty chooser
            if Game.difficulty:
                Game.draw_text('Choose Difficulty', (Game.WIDTH / 2, Game.HEIGHT / 2 - 200), Game.header_font)
                Game.normal_btn.render()
                Game.hard_btn.render()
                Game.extreme_btn.render()

            # in scoreboard room
            if Game.score_board:
                Game.draw_text('Score Board', (Game.WIDTH / 2, Game.HEIGHT / 2 - 200), Game.header_font)
                sorted_scores = dict(sorted(Game.scores.items(), key=lambda item: item[1]))

                off = -100
                for key in sorted_scores.keys():
                    if sorted_scores[key] != 0:
                        Game.draw_text(key, (Game.WIDTH / 2 - 100, Game.HEIGHT / 2 + off), Game.header_font)
                        Game.draw_text(str(sorted_scores[key]),
                                       (Game.WIDTH / 2 + 100, Game.HEIGHT / 2 + off), Game.header_font)
                        off += 100

            # when starting the game
            if Game.game_start and Game.frame_cycles - Game.begin_cycles < 3:
                begin_text = Game.header_font.render('Begin!', True, Game.COLOR_BLACK)
                Game.surface.blit(begin_text, (Game.WIDTH / 2 - begin_text.get_width() / 2,
                                               Game.HEIGHT / 2 - begin_text.get_height() / 2))

        Game.screen.blit(Game.surface, (0, 0))
        pyg.display.flip()

class Comm(Thread):
    """
    A thread to communicate with the server and deliver messages
    """
```

```python
def run(self) -> None:
    while Game.is_running:
        msg = Game.csocket.recv(SIZE).decode()
        if msg == DIFF:  # server asking to choose difficulty
            Game.difficulty = True
        elif msg.startswith(BEGIN):  # the game shall start
            Game.game_start = True
            Game.begin_cycles = Game.frame_cycles
            Game.csocket.send(ACK.encode())
            Game.next_level()  # exit lobby level
            Game.levels = Game.levels[:int(msg.split(':')[1]) + 2]  # adjust level amount to difficulty
        elif msg.startswith(SCORE):  # receive scores to show on leaderboard
            msg_split = msg.split(':')[1:]
            for i in range(0, len(msg_split) - 1, 2):
                if msg_split[i] != Game.player1.color:
                    Game.scores[msg_split[i]] = int(msg_split[i + 1])

            Game.csocket.send(f'{Game.frame_cycles}'.encode())
        elif msg.startswith(POS):  # receive position of other players
            dic = {}
            data = msg.split(':')[1:]
            for i in range(0, len(data) - 1, 2):
                data_list = data[i + 1][1:-1].split(', ')
                dic[data[i]] = (int(data_list[0]), float(data_list[1]), float(data_list[2]))

            for color in dic.keys():
                if Game.opponents[color] != Game.player1:
                    Game.opponents[color].level = dic[color][0]
                    Game.opponents[color].x = dic[color][1]
                    Game.opponents[color].y = dic[color][2]
```

```python
        # when in scoreboard room, send indication and current position for shared lounge effect
        if Game.current_level == len(Game.levels) - 1:
            Game.csocket.send(f'{DONE}:{Game.current_level}:{Game.player1.x}:{Game.player1.y}'.encode())
            for p in Game.opponents.keys():
                if Game.opponents[p] == Game.player1:
                    Game.scores[p] = Game.frame_cycles
                    break
        else:  # send current position in level to other players
            Game.csocket.send(f'{Game.current_level}:{Game.player1.x}:{Game.player1.y}'.encode())

    Game.csocket.send(QUIT.encode())
```

```python
def track_player(self, player):
    """
    Communicate with specific player (threaded)
    """
    player.begin(self.difficulty)

    while True:
        try:
            # compose a message containing all positions of players
            pos_msg = ''
            for key in self.positions.keys():
                pos_msg += f':{key}:{self.positions[key]}'
            player.send(f'{POS}{pos_msg}')
            data = player.recv()

            if data.startswith(DONE):
                # compose a message containing all scores of players
                score_msg = ''
                for key in self.scores.keys():
                    score_msg += f':{key}:{self.scores[key]}'
                player.send(f'{SCORE}{score_msg}')
                score = int(player.recv())

                # receive player score and location in lounge
                self.scores[player.color] = score
                split = data.split(':')[1:]
                self.positions[player.color] = (int(split[0]), float(split[1]), float(split[2]))
                continue
            elif QUIT in data:
                player.socket.close()
                break

            # update positions
            data_list = data.split(':')
            self.positions[player.color] = (int(data_list[0]), float(data_list[1]), float(data_list[2]))
```

הסבר

קטעי הקוד האלו מייצגים את שני צדדי התקשורת המרכזיים, צד השחקן וצד החדר המפעיל את המשחק. בקטעים ניתן לראות כיצד השחקן מעדכן את החדר בכל פעולותיו ומיקומיו ואת החדר מעדכן אותו בחזרה בפעולותיהם ומיקומיהם של שחקנים אחרים בחדר. בכל הנוגע לציור השחקנים בחלון המשחק, Thread הציור מוודא שכל המצויר אכן שייך לחלון ולשלב הרלוונטי, ושהשחקן המרכזי אותו אנו משחקים תמיד יבלוט ויהיה בראש הערימה.

הסבר פעילות שחקן

```python
import game
import game_objects
import obstacles
import power_ups


class Player(game_objects.GameObject):
    """
    Player class, sets the actual player, and the "shadows" (other players on the game)
    """

    def __init__(self, color) -> None:
        super().__init__(
            game.Game.TYPE_PLAYER, color,
            game.Game.start_points[game.Game.current_level][0],
            game.Game.start_points[game.Game.current_level][1], 40, 40
        )

        self.vx: float = 0
        self.vy: float = 0

        self.is_standing: bool = False
        self.is_floating: bool = False
        self.is_jumping: bool = False
        self.is_alive: bool = True
        self.keep_jumping: bool = False

        self.total_keys_collected: int = 0

        self.before: tuple = ()
        self.s_modifier: float = -10
        self.s_direction: int = 1
        self.level = 0

    def reset(self) -> None:
        """
        On contact with obstacle, reset the level
        """
        self.x = game.Game.start_points[game.Game.current_level][0]
        self.y = game.Game.start_points[game.Game.current_level][1]

        self.vx = 0
        self.vy = 0

        self.is_standing = False
        self.is_floating = False
        self.is_jumping = False
        game.Game.is_floating = False

        self.total_keys_collected = 0

        self.before = ()
        self.s_modifier = -10
        self.s_direction = 1
```

```python
def passed(self) -> bool:
    """
    Checks if player exits the "boundaries" (the screen size)
    """
    return self.x + self.width <= 0 or self.x >= game.Game.WIDTH or \
           self.y + self.height <= 0 or self.y >= game.Game.HEIGHT

def die(self) -> None:
    """
    Kills player, unless is in God Testing Mode
    """
    if not game.Game.god_mode:
        tvx: float = self.vx
        self.reset()
        self.vx = tvx

        self.is_alive = False

def fall(self) -> None:
    """
    Attract player towards ground
    """
    if self.is_floating:
        if self.vy > -1:
            self.vy += -0.02
    else:
        self.vy += game.Game.GRAVITY

def jump(self) -> None:
    """
    Apply vertical velocity upwards to simulate player jumping
    """
    if self.is_standing:
        self.vy = -5
        self.is_jumping = True
        self.keep_jumping = True
    elif self.is_jumping:
        if -8.5 < self.vy < 0 and self.keep_jumping:
            self.vy -= 1
        else:
            self.keep_jumping = False

def float(self) -> None:
    """
    On interaction with "Jet", levitate and float of the ground, ignoring gravity
    """
    self.before = (game.Game.frame_cycles, game.Game.frame_count)
    self.is_floating = True

    if self.vy < 0:
        self.vy *= 0.2
    else:
        self.vy = 0
```

```python
def start_left(self) -> None:
    """
    Start movement to the left
    """
    self.vx = -3

def start_right(self) -> None:
    """
    Start movement to the right
    """
    self.vx = 3

def stop_left(self) -> None:
    """
    Stop movement to the left
    """
    if self.vx == -3:
        self.vx = 0

def stop_right(self) -> None:
    """
    Stop movement to the right
    """
    if self.vx == 3:
        self.vx = 0

def check_miscellaneous(self, b) -> bool:
    """
    Check for intersection with miscellaneous obstacles
    """
    if b.type == game.Game.TYPE_CROSSBOW:
        if self.is_colliding(b.arrow.get_rect()):
            self.die()
            return True

    if b.is_colliding(self.get_rect()):
        if b.__class__.__bases__[0] == power_ups.PowerUp:
            if not b.collected:
                b.collect()

                if b.type == game.Game.TYPE_KEY:
                    self.total_keys_collected += 1
                elif b.type == game.Game.TYPE_JET:
                    self.float()
                return True
            else:
                return True

        elif b.__class__.__bases__[0] == obstacles.Obstacle:
            self.die()
            return True

        elif b.type == game.Game.TYPE_DOOR:
            if b.is_open:
```

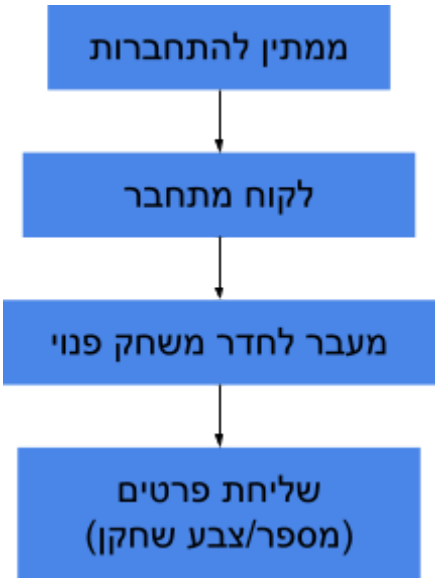
```
                return True

    return False

def update_pos(self):
    """
    Update position according to velocity
    """
    self.x += self.vx
    self.y += self.vy

def update(self) -> None:
    """
    Update the player state in this frame
    """
    if game.Game.is_rotating:
        self.vy -= game.Game.GRAVITY
        return

    if self.is_floating:
        if (game.Game.frame_cycles * 60 + game.Game.frame_count) - (self.before[0] * 60 + self.before[1]) >= 240:
            self.is_floating = False

    self.update_pos()

    prev_x: float = self.x - self.vx
    prev_y: float = self.y - self.vy

    self.is_standing = False

    # check intersections on vertical axis
    for b in game.Game.get_level_objects():
        if self.check_miscellaneous(b):
            continue

        if b.is_colliding(self.get_rect()):
            if self.vy < 0 and prev_y >= b.y + b.height:
                if prev_x == b.x + b.width or prev_x + self.width == b.x:
                    continue

                self.y = b.y + b.height
                self.vy = 0.5
            elif self.vy > 0 and prev_y + self.height <= b.y:
                if prev_x == b.x + b.width or prev_x + self.width == b.x:
                    continue

                self.y = b.y - self.height

                if b.type == game.Game.TYPE_SPRINGBOARD:
                    if self.vy < 15:
                        self.vy *= -1.1
                    else:
                        self.vy *= -1
                else:
```

```
                    self.vy = 0
                    self.is_standing = True
                    self.is_jumping = False

    # check intersections on horizontal axis
    for b in game.Game.get_level_objects():
        if self.check_miscellaneous(b):
            continue

        if b.is_colliding(self.get_rect()):
            if self.vx < 0 and prev_x >= b.x + b.width:
                if self.is_standing and self.y == b.y:
                    continue

                self.x = b.x + b.width

                if self.vy > 0 and not self.is_floating:
                    self.vy -= game.Game.GRAVITY / 2
            elif self.vx > 0 and prev_x + self.width <= b.x:
                if self.is_standing and self.y == b.y:
                    continue

                self.x = b.x - self.width

                if self.vy > 0 and not self.is_floating:
                    self.vy -= game.Game.GRAVITY / 2

    if self.vx == 0 and self.vy == 0:
        self.s_modifier += 0.2 * self.s_direction
        if self.s_modifier > 10 or self.s_modifier < -10:
            self.s_direction *= -1
    else:
        self.s_modifier = -10

def render(self, surface, pyg) -> None:
    """
    Render the player to the screen
    """
    pyg.draw.rect(surface, self.color, self.get_rect())
    pyg.draw.rect(surface, game.Game.COLOR_BLACK, self.get_rect(), 2)
```

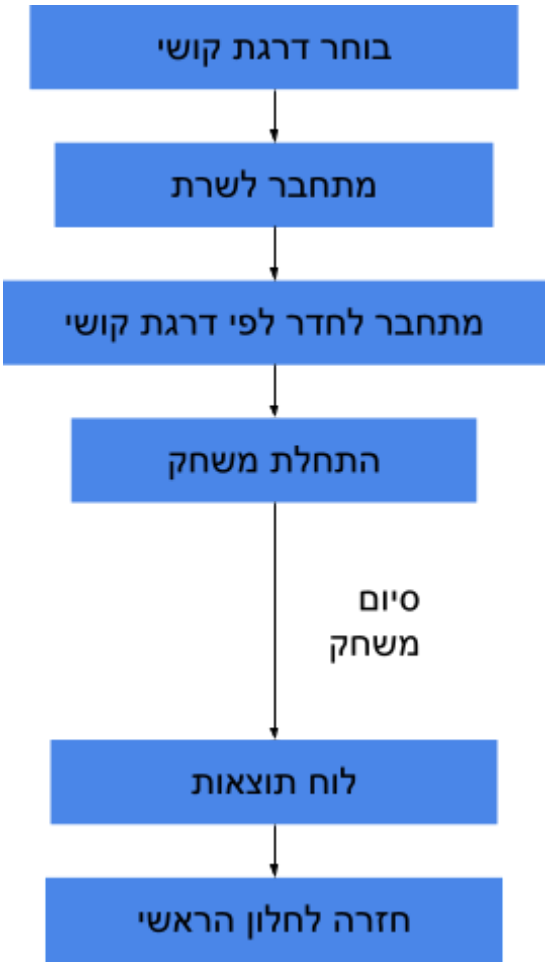
הסבר

מחלקה זו מייצגת את השחקן ומנהלת את הלוגיקה הדרושה בכדי לתקשר עם שאר חלקי המשחק בצורה המצופה מן המשתמש. מחלקה זו מגיבה לפעולות המקלדת של המשתמש ובודקת אינטראקציה עם גופים על המסך לפיה קובעת המשך התנהגות.

פרוטוקול שרת



פרוטוקול לקוח



תיאור המחלקות מרכזיות

Game Class

`def draw_text(text, pos, font):`	`Draw centered text to surface`
`def initialize() -> None:`	`Initialize elements and libraries`
`def rotate(rd: int) -> None:`	`Rotate the level`
`def stop_rotation() -> None:`	`Stop rotating the level`
`def run() -> None:`	`Run the main game loop`
`def stop_running() -> None:`	`Stops running the game`
`def observe_welcome_events() -> None:`	`Catch events on the welcome screen`
`def observe_events() -> None:`	`Catch events on game screen`
`def update_objects() -> None:`	`Update all game objects in level`
`def add_level(lvl: tuple) -> None:`	`Adds level`
`def new_start_point(x: float, y: float) -> None:`	`Creates player start point`
`def next_level() -> None:`	`Transition to the next level`
`def get_level_objects() -> tuple:`	`Return level objects`

`def generate_levels() -> None:`	`Build and list all levels in the game`
	`Lobby: 0`
	`Normal: 1st -> 3rd`
	`Hard: 1st -> 5th`
	`Extreme: 1st -> 6th`
	`Score Lounge: 7th`

`def init_level() -> None:`	`Initialize keys for current level`

Room Class

`def run(self):`	*Request difficulty and start game*
`def play(self):`	*Start threads for players, then wait for them to finish*
`def track_player(self, player):`	*Communicate with specific player (threaded)*
`def add_player(self, csocket):`	*Add a player to the room*
`def is_full(self):`	*Check if room is full*

Player Class

`def reset(self) -> None:`	*On contact with obstacle, reset the level*
`def passed(self) -> bool:`	*Checks if player exits the "boundaries" (the screen size)*
`def die(self) -> None:`	*Kills player, unless is in God Testing Mode*
`def fall(self) -> None:`	*Attract player towards ground*
`def jump(self) -> None:`	*Apply vertical velocity upwards to simulate player jumping*
`def float(self) -> None:`	*On interaction with "Jet", levitate and float of the ground, ignoring gravity*
`def start_left(self) -> None:`	*Start movement to the left*
`def start_right(self) -> None:`	*Start movement to the right*
`def stop_left(self) -> None:`	*Stop movement to the left*
`def stop_right(self) -> None:`	*Stop movement to the right*
`def check_miscellaneous(self, b) -> bool:`	*Check for intersection with miscellaneous obstacles*
`def update_pos(self):`	*Update position according to velocity*
`def update(self) -> None:`	*Update the player state in this frame*

`def render(self, surface, pyg) -> None:`	`Render the player to the screen`

חוקי המשחק

תיאור

כל שחקן מיוצג על ידי ריבוע נושם על המסך (הדמות). את הדמות שלו יוכל השחקן להזיז באמצעות מקשי החצים ברגע שיתחיל המשחק. למשחק שלבים שונים וכולם מצוירים באותו גודל בחלון המשחק. מטרת השחקנים הינה לזוז ברחבת השלב ולתקשר עם חלקי המשחק השונים. בכדי לעבור שלבים ולהתקדם במשחק יהיה על השחקנים לאסוף מפתחות צהובים, מפתחות אלו יפתחו את הדלת לשלב הבא אליה יוכל להיכנס השחקן רק בתנאי שאסף את כל המפתחות המפוזרים באותו השלב.

מסלול וניצחון

כל השחקנים מתחילים באותה נקודה ומשם מתחילים להתחרות על זמן, מי שמגיע לסוף בזמן הכי קצר מנצח ומוכתר אלוף המסלול. השחקנים רואים אחד את השני בזמן המשחק כך שהתחרותיות מוגברת ואינטנסיבית. בכל שלב יש חלקים שונים המבצעים פעולות שונות, לדוגמא, ישנה מקפצה המקפיצה את השחקן לגובה מסוים בדרך כלל כדי שיוכל להגיע לפלטפורמה גבוהה. או, מצד שני, יכולת המאפשר לשחקן לרחף לכמה שניות ובכך להשיג יתרון.

סוף המשחק

כל שחקן המגיע לסוף המסלול או מוותר על השתתפותו מגיע ללוח התוצאות ומחכה לשאר השחקנים שיגיעו. ברגע שכל השחקנים הגיעו, לוח התוצאות אז שלם ומוכן להכריז על האלוף המכריע של אותו משחק. לאחר ההכרזה, נגמר המשחק וחוזרים לחלון הראשי עם אפשרות להתחיל מחדש.