Fachinformatiker für Anwendungsentwicklung

Dokumentation einer schulischen Projektarbeit
Thema:
Erstellung eines simplen Transporterspiels

Pruefling:
Pascal Raber

INHALTSVERZEICHNIS

Inhaltsverzeichnis

1. Einleitung
    a. Projektziel

        Das Ziel des Projekts ist die Entwicklung eines 2D-Spiels, in dem der Spieler einen LKW steuert, um Erz zu transportieren, während er Ressourcen wie Treibstoff verwaltet und sich vor einem Helikopter schützt, der versucht, das Erz zu stehlen. Das Spiel soll grundlegende Mechaniken wie Bewegung, Kollisionserkennung, Ressourcenmanagement und Sieg-/Niederlagenbedingungen implementieren.

    b. Ausgangssituation

        Das Projekt wurde im Rahmen des Berufsschulunterrichts im Lernfeld 8 als Probegang für das Abschlussprojekt zur Ausbildung zum Fachinformatiker für Anwendungsentwicklung durchgeführt. Ziel ist es, die im Laufe der Ausbildung erworbenen Kenntnisse in den Bereichen objektorientierte Programmierung, Softwarearchitektur und Spieleentwicklung praktisch anzuwenden und ein Projekt im Rahmen des Berufsschulunterrichts durchzugehen und zu üben.

    c. Projektumfang
        Das Spiel umfasst folgende Hauptfunktionen:

        - Spielersteuerung: Der Spieler kann den LKW mit den Tasten W, A, S, D oder den Pfeiltasten bewegen.
        - Helikopter-KI: Der Helikopter verfolgt den LKW und versucht, Erz zu stehlen.
        - Ressourcenmanagement: Der Spieler muss den Treibstoff des LKWs überwachen und an einer Tankstelle auffüllen.
        - Kollisionserkennung: Interaktionen zwischen dem LKW, dem Helikopter und stationären Objekten wie der Tankstelle, dem Lager und dem Zuhause werden erkannt und verarbeitet.
        - Sieg-/Niederlagenbedingungen: Der Spieler gewinnt, wenn eine bestimmte Menge Erz im Lager gespeichert ist, und verliert, wenn der Helikopter zu viel Erz stiehlt.

2. Projektplanung
    a. Zeitlicher Ablauf

Phase	Aufgabe	Dauer (Stunden)
1. Planung		12 Stunden
	Analyse der Anforderungen (funktional und nicht-funktional)	4 Stunden
	Erstellung eines UML-Klassendiagramms	3 Stunden
	Erstellung eines Zeitplanung	2 Stunden
	Auswahl der Technologien und Werkzeuge	3 Stunden
2. Implementierung		50 Stunden
	Einrichtung der Entwicklungsumgebung	2 Stunden
	Implementierung der Klasse GameMGMT	10 Stunden
	Implementierung der Klasse Truck	8 Stunden
	Implementierung der Klasse Helicopter	8 Stunden
	Implementierung der statischen Objekte (Home, GasStation, Storage)	8 Stunden
	Implementierung der Spielmechaniken (Bewegung, Kollisionen, Sieg/Niederlage)	10 Stunden
	Integration der Benutzeroberfläche (Textanzeige, Eingabefelder, Buttons)	4 Stunden
3. Peer Reviews		8 Stunden
	Durchführung von Peer Reviews (Code- und Funktionsüberprüfung)	8 Stunden
4. Dokumentation		9 Stunden
	Erstellung der Projektdokumentation (Einleitung, Planung, Umsetzung, Fazit)	9 Stunden
5. Abschluss		1 Stunde
	Präsentation und Reflexion	1 Stunde

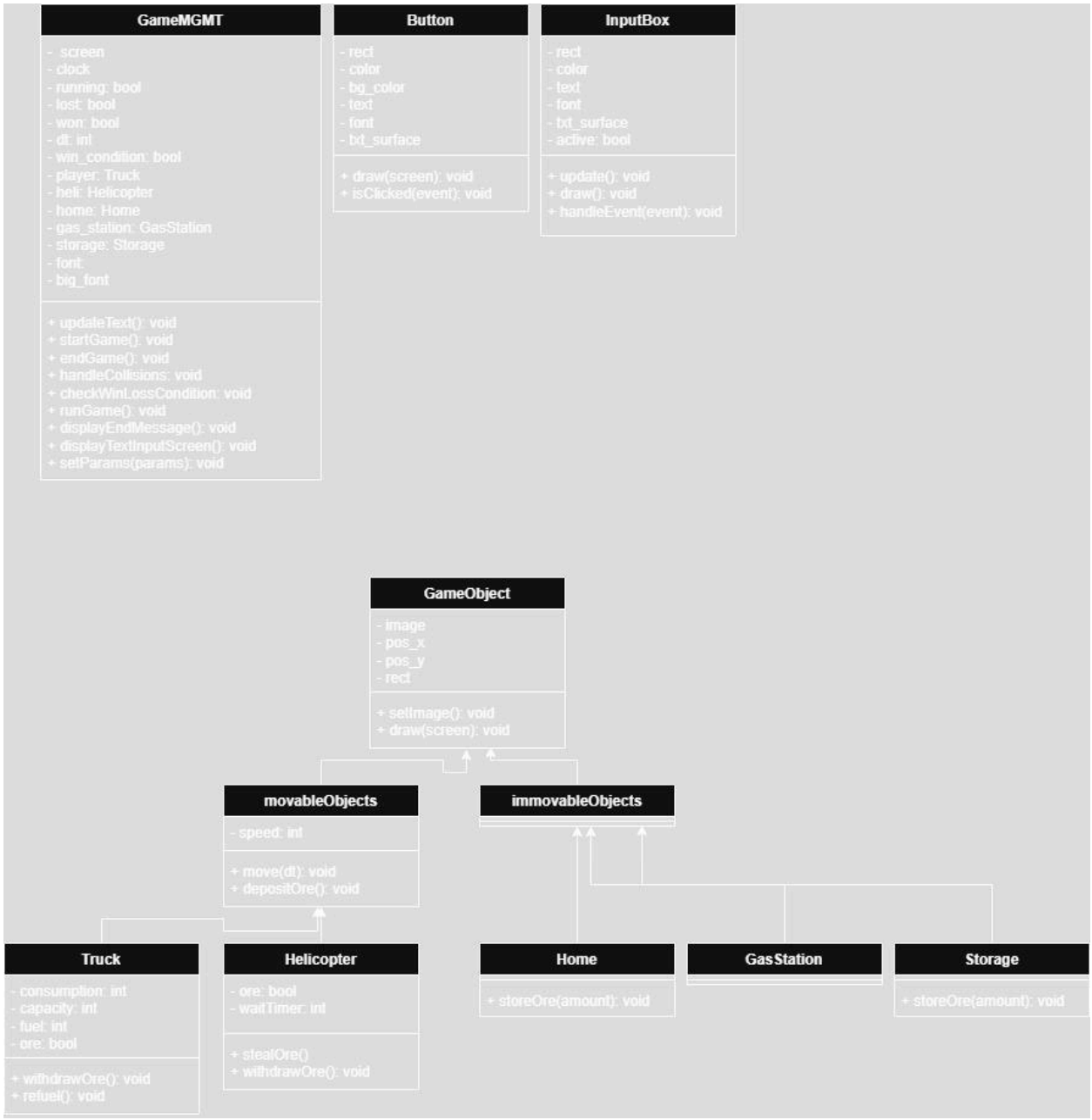
Tabelle 1: Zeitplan

    b. Benutzte Werkzeuge

Programmiersprache: Python
Framework: Pygame
Entwicklungsumgebung: Visual Studio Code
Versionskontrolle: GitHub
Diagrammerstellung: Draw.io

3. Projektumsetzung
    a. Zielsetzung
    Zielsetzung des Projektes war es ein Transporterspiel in Python (mit Hilfe von dem Pygame Framework) zu erstellen um einerseits die Erstellung von einem IHK Projekt an einem richtigen Projekt zu proben und andererseits das Programmieren in Python nochmal verstärkt zu lernen.
    b. Anforderungen
    Die Anforderungen waren wie folgt (Auszug aus der praesentation zum Kick off des Projektes):
    Pythonspiel mit Grafik (Bibliothek „pygame“), lauffähig auf Linux-PC mit Python, optional auf Android-Smartphone z. B. mit Pydroid
    ● Benutzerdokumentation (Systeminstallation, Interaktion)
    ● Programmdokumention (UML-Diagramme, Systemarchitektur, Struktogramme)
        - Programm in Python mit sauberem Programmcode
        - kein unnötiger Typwechsel von Variablen
        - Variablen, Methoden und Klassen gut dokumentiert

    c. Architektur



- GameMGMT
  Zentrale Klasse zum Managen der ganzen Parameter und GameObjects. Hier ist auch der Gameloop enthalten.

- GameObject
  Zentrale Klasse, die als Basis von movableObjects und immovableObjects gilt. Hier werden einige Attribute und Methoden für die Kind- und Enkelklassen vorgegeben.

- MovableObjects & immovableObjects

Zentrale Klassen die als Basis von Truck, Helicopter, Home, GasStation und Storage gelten. Hier werden auch wieder Attribute und Methoden für die Kindklassen vorgegeben.

- Truck
  Spielerklasse.

- Helicopter
  Gegnerklasse.

- Home, Storage & GasStation
  Statische Klassen, die Erz “beherbergen“ können (Home & Storage), oder es ermöglichen, den Treibstoff des Autos wieder aufzufüllen.

d. Implementierung
   - Bewegung des Trucks
     Die Bewegung des Trucks ist sehr simpel aufgebaut.
     Nach einer Überprüfung auf den Tankstand des Trucks wird das Modell des Trucks wird, je nachdem welcher input getätigt wird, in die jeweilige Richtung bewegt.

```
def move(self, dt):
    if self.m_fuel > 0:
        keys = pygame.key.get_pressed()
        if keys[pygame.K_w] or keys[pygame.K_UP]:
            self.m_pos_y -= self.m_speed * dt
            self.consume_fuel(dt)
        if keys[pygame.K_s] or keys[pygame.K_DOWN]:
            self.m_pos_y += self.m_speed * dt
            self.consume_fuel(dt)
        if keys[pygame.K_a] or keys[pygame.K_LEFT]:
            self.m_pos_x -= self.m_speed * dt
            self.consume_fuel(dt)
        if keys[pygame.K_d] or keys[pygame.K_RIGHT]:
            self.m_pos_x += self.m_speed * dt
            self.consume_fuel(dt)
        self.m_rect.center = (self.m_pos_x, self.m_pos_y)
    else:
        return True
```

   - Bewegung des Helicopters
     Der Helikopter bewegt sich grundsaetzlich genauso wie der Truck, nur anstatt vom Spieler direkt gesteuert zu werden, bewegt sich der Helikopter auf die Position des Spielers, wenn der Helikopter kein Erz hat.

Wenn der Helikopter aber schon Erz geklaut hat, bewegt sich der Helikopter direkt zum nächsten Bildschirmrand und verschwindet außerhalb des Spielbildschirms. Dort wartet der Helikopter 30 Sekunden, bevor er das verfolgen des Spielers wieder aufnimmt.

```python
def move(self, dt, truck_pos_x, truck_pos_y):
    if self.m_ore:
        if self.m_pos_y > -self.m_rect.height:
            self.m_pos_y -= self.m_speed * dt
        else:
            self.m_wait_timer += dt
            if self.m_wait_timer >= 30:  # Wait for 30 seconds complete
                self.m_ore = False
                self.m_wait_timer = 0
                self.m_pos_y = 0  # Reset position to re-enter the screen
    else:
        if self.m_pos_x < truck_pos_x:
            self.m_pos_x += self.m_speed * dt
        if self.m_pos_x > truck_pos_x:
            self.m_pos_x -= self.m_speed * dt
        if self.m_pos_y < truck_pos_y:
            self.m_pos_y += self.m_speed * dt
        if self.m_pos_y > truck_pos_y:
            self.m_pos_y -= self.m_speed * dt

    self.m_rect.center = (self.m_pos_x, self.m_pos_y)
```

- Collision Handling
  Collision Handling wird in GameMGMT zentral gemacht

```python
def handle_collisions(self):
    """
    Handle collisions between the player and other objects.
    """
    if self.m_player.m_rect.colliderect(self.m_gas_station.m_rect):
        self.m_player.refuel()

    if self.m_player.m_rect.colliderect(self.m_home.m_rect) and not self.m_player.m_ore and not self.m_home.m_ore_stored == 0:
        self.m_player.withdraw_ore()
        self.m_home.store_ore(self.m_player.m_capacity)

    if self.m_player.m_ore:
        if self.m_player.m_rect.colliderect(self.m_storage.m_rect):
            self.m_storage.store_ore(self.m_player.m_capacity)
            self.m_player.deposit_ore()

        if self.m_player.m_rect.colliderect(self.m_heli.m_rect) and not self.m_heli.m_ore:
            self.m_heli.steal_ore(self.m_player.m_capacity)
            self.m_player.deposit_ore()
```

- Win/ Loss Konditionen
  Wird auch zentral in GameMGMT gemacht

```python
    def check_win_loss_conditions(self):
        """
        Check if the player has won or lost the game.
        """
        if self.m_storage.m_ore_stored >= self.m_win_condition:
            self.m_won = True
            self.end_game()

        if self.m_heli.m_ore_stolen > 100 - self.m_win_condition:
            self.m_lost = True
            self.end_game()
```

4. Testphase
    a. Peer Reviews
    Die Testphase wurde durch Peer Reviews von Mitschülern durchgeführt.
    Hier wurden 2 bis 3 Schüler zufällig ausgewählt und hatten den Auftrag das Spiel auf kleinste Details zu testen. Diese sind mit 95% Erfolgsrate zurückgekommen und es wurden keine massiven Fehler erkannt.

5. Fazit
    Dieses Projekt lief einigermaßen reibungslos ab, was an der vorhandenen Programmiererfahrung und der Nicht-Komplexität des Projektes lag. Die Peer Reviews hätten gerne anders laufen können, da hier auch bei Fehlern kein wirkliches Feedback kam, leider. Alles in allem lief das Projekt aber ziemlich gut.

6. Anhaenge