Fortgeschrittene Programmierung:
Abschlussdokumentation „Run & Boom“ - Team 04

Konzept: Spielidee & Spielmechanik

Titel: „Run & Boom“

Das Spiel „Run & Boom“ zielt im Allgemeinen auf einen Wettkampf zwischen zwei Spielern ab, wobei kurze und intensive Runden im Vordergrund stehen. Dabei ist von den jeweiligen Spielern sowohl Präzision und Reaktionsschnelligkeit als auch Ausweichen und Zielen gefordert.

Im Duell treten zwei Spieler gleichzeitig gegeneinander an, wobei eine einzige Tastatur gemeinsam als Eingabegerät genutzt wird. Hierbei verwendet der Spieler 1 die Tasten: W, A, S, D und Spieler 2 die vier Pfeiltasten zur Eingabe.
Zur Unterscheidung sind die Spielfiguren der zwei Spieler farblich gekennzeichnet. Spieler 1 steuert die blaue Figur „Blue“, wohingegen Spieler 2 die rote Figur „Red“ bedient.

In der ersten Spielrunde übernimmt Spieler 1 mit „Blue“ die Rolle des Runners und bewegt sich in Richtung des rechten Bildschirmrandes, mit dem Ziel, einen Checkpoint zu erreichen. Spieler 2 hingegen steuert gleichzeitig „Red“ in der Rolle einer Kanone, die am rechten Bildschirmrand fixiert ist und sich lediglich in vertikale Richtung hoch und runter bewegen lässt. Darüber hinaus ist für Spieler 2 durch Drücken der linken Pfeiltaste das Abfeuern von Projektilen in Richtung des Runners „Blue“ möglich.

Ziel jeder Runde ist es, dass der Runner den Checkpoint erreicht, bevor er von den Kanonenprojektilen getroffen wird. Auf dem Weg zum Checkpoint tauchen jedoch unerwartet Hindernisse für den Runner auf, denen er ausweichen muss. Im Gegenzug bieten ihm diese Hindernisse aber auch Schutz vor anfliegenden Projektilen der Kanone.

Nach jedem Durchgang, unabhängig von der Eliminierung des Läufers durch Kanonenschüsse oder dem Erreichen des Checkpoints, findet ein Rollenwechsel der Spieler statt: So treten in der darauffolgenden Runde Spieler 1 mit „Blue“ als Kanone und Spieler 2 mit „Red“ als Runner gegeneinander an.

Punktesystem:

Das Score-System stellt ein kompetitives Runden- und Satz-System dar, bei dem sowohl der Runner als auch die Kanone aktiv punkten können.

Die Punktevergabe innerhalb einer Runde erfolgt für den Runner, wenn dieser den Checkpoint erreicht. Die Kanone hingegen erzielt einen Punkt, wenn sie den Runner abschießt oder dieser aufgrund von Hindernissen über den linken Bildschirmrand gedrängt wird. Unabhängig vom Ausgang der Runde findet anschließend ein sofortiger Rollenwechsel zwischen Runner und Kanone statt.

Die Punktevergabe innerhalb eines Satzes erfolgt durch addierte Rundenpunkte der Spieler. Sobald ein Spieler 3 Runden (unabhängig, ob diese Punkte als Runner oder als Kanone erzielt wurden) für sich entscheiden konnte, gewinnt er den aktuellen Satz (Standardeinstellung: 3 Rundenpunkte = 1 Satzpunkt).

Das übergeordnete Ziel ist der Gesamtsieg eines Spiels. Das Spiel endet, sobald ein Spieler insgesamt 3 Sätze gewonnen hat (Standardeinstellung: 3 Satzpunkte = gesamter Spielgewinn). Ab diesem Zeitpunkt wird keine neue Runde mehr begonnen und der Siegerbildschirm wird angezeigt.

Ein wesentliches Merkmal dieses Punktesystems ist der Reset der Rundenpunkte nach einem Satzgewinn. Somit wird der Runden-Punktestand beider Spieler beim Beginn eines neuen Satzes jeweils auf 0 gesetzt. Dadurch beginnt der Kampf um den nächsten Satzpunkt immer wieder aus einer neutralen Ausgangslage.

Darüber hinaus ist anzumerken, dass sowohl die Gewinnrunden (wie viele Runden führen zu einem Satzpunkt?) als auch die Gewinnsätze (wie viele Sätze führen zum Gesamtsieg?) mit Leichtigkeit im Programmcode des Spiels angepasst werden können. Standardmäßig gilt: 3 Rundenpunkte ergeben 1 Satzpunkt und 3 Satzpunkte haben den gesamten Spielgewinn zur Folge.

Dieses Spielkonzept erfordert von den Spielern starkes Reaktionsvermögen, gutes Movement als Runner und exaktes Timing beim Abfeuern der Projektile, aber auch gegenseitiges Vorhersehen der Handlungen.

Steuerung und Anleitung:

Durch den ständigen Rollenwechsel von Runner und Kanone sind im Folgenden jeweils zwei Tastenbelegungen einer Steuerung zugewiesen: Eine Taste für den Fall, wenn Spieler 1 (W, A, S, D) mit „Blue“ die Rolle übernimmt, und eine Steuerung, wenn Spieler 2 (4 Richtungen der Pfeiltasten) mit „RED“ die Rolle übernimmt.

Steuerung des Runners:

Die Bewegungssteuerung des Runners beinhaltet:

- Bewegung zurück (nach links): Spieler 1: A | Spieler 2: Pfeiltaste links
- Bewegung vorne (nach rechts): Spieler 1: D | Spieler 2: Pfeiltaste rechts
- Bewegung nach oben (Wechsel in die obere Ebene):
  Spieler 1: W | Spieler 2: Pfeiltaste hoch
- Bewegung nach unten (Wechsel in die untere Ebene):
  Spieler 1: S | Spieler 2: Pfeiltaste runter

Steuerung der Kanone:

Die Bewegungssteuerung der Kanone beinhaltet:

- Bewegung nach oben (Wechsel in die obere Ebene):
  Spieler 1: W | Spieler 2: Pfeiltaste hoch

- Bewegung nach unten (Wechsel in die untere Ebene):
  Spieler 1: S | Spieler 2: Pfeiltaste runter
- Abfeuern eines Kanonenprojektils in Richtung des Runners:
  Spieler 1: A | Spieler 2: Pfeiltaste links

Allgemeine Steuerung:

- Im Startmenü: Die Namenseingabe im Startmenü ist optional. Hierbei besteht die Möglichkeit, Spieler 1 und Spieler 2 umzubenennen. Mit der Tabulator-Taste (Tab) wird das Eingabefeld zur Umbenennung des Spielers 2 gewechselt. Wenn keine Eingabe stattfindet, werden die Standardnamen (Default-Namen) „Spieler 1“ und „Spieler 2“ verwendet.

- Im Menü: Als Bestätigung zum Spielstart und zum Beginnen der nächsten Spielrunde ist das einmalige Drücken der Eingabe vorgesehen.

- Darüber hinaus kann das Spiel jederzeit über die Escape-Taste (Esc) oder das Schließen des Spielefensters beendet werden.

Spielmechanik:

Das Spielfeld scrollt automatisch in einer definierten Geschwindigkeit von rechts nach links und zwingt somit den Runner, sich in Richtung des Checkpoints zu bewegen. Dabei kann sich der Runner nur bis kurz vor die Kanone frei im sichtbaren Bereich aufhalten. Dies ist aber aufgrund der dadurch verkürzten Distanz zur Kanone und geringerer Reaktionszeit zum Ausweichen der Projektile nachteilig. Ebenso darf der Runner den linken Bildschirmrand nicht verlassen. Wird er so weit nach links aus der Welt gedrängt, dass er nicht mehr sichtbar ist, endet der Durchgang mit Punkt für die Kanone.

Das Spielfeld besteht aus acht festen Ebenen, zwischen denen der Runner und die Kanone frei wechseln können. Jedoch gibt es eine minimale Cooldown-Zeit, bevor in die nächste Ebene gewechselt werden darf (switch-lane-speed). Ebenso wurde eine geringe Nachladezeit (reload-time) nach jedem Schuss der Kanone eingeführt. Zudem wirken Kanonenprojektile nur bei direktem Treffer des Runners und verpuffen bei Kontakt mit Hindernissen.

Darüber hinaus erscheinen zwei verschiedene Arten von Hindernissen in den Ebenen, die von dem rechten Bildschirmrand in das Level geflogen kommen: relativ kurze und längere Grasblöcke. Die Entscheidung, in welcher Ebene welches Hindernis in welcher Größe als Nächstes auftaucht, ist randomisiert und völlig zufällig, wobei keine Hindernisse direkt hintereinander in einer Ebene spawnen können. Diese Mechanik wurde zum Schutz vor Bugs und als Prävention von übereinanderliegenden Hindernissen eingeführt.

Durch das zufällige Generieren der Hindernisse wird dafür gesorgt, dass wirklich jedes Level einzigartig ist und sich von den zuvor gespielten Leveln in der Hindernis-Anordnung unterscheidet.

Bei einer frontalen Kollision zwischen einem Hindernis und dem Runner wird dieser durch den automatischen Scroll-Speed der Welt kontinuierlich nach links geschoben und ein Ausweichen in eine andere Ebene ist nötig.

Der Checkpoint erscheint nach einer festgelegten Distanz bzw. Laufzeit der Runde und gilt als Ziel, das der Läufer erreichen muss, um einen Rundenpunkt zu erhalten.

Durch dieses Zusammenspiel aus mehreren Ebenen, anfliegenden Hindernissen, dem Rollenwechsel von Runner und Kanone sowie dem automatischen Bildlauf entsteht ein dynamisches Spiel, in dem Reaktionsfähigkeit, Mustererkennung und Timing für den Sieg entscheidend sind.

Im Laufe der Konzeptionierung und Implementierung lag ein großer Fokus auf der Anpassung von Parametern und dem „Balancing“ zwischen Runner und Kanone. Mit dem Ziel, ein simples, kompetitives, aber dennoch faires Spiel zu entwickeln wurde viel Zeit in ausgiebigen Testläufen mit Anpassungen von entscheidenden Variablen verbracht. Beispielsweise wurden Veränderungen an der Scroll-Geschwindigkeit des Levels (scroll-speed), der Runner-Geschwindigkeit (runner- acceleration und runner-friction), der Geschwindigkeit zum Wechseln von Ebenen (switch-lane-speed), der Nachladezeit der Kanon (reload-time) und an der Anzahl von Hindernissen (spawn-rate) pro Level vorgenommen.

Story hinter „Run & Boom“: [**1) KI-Hinweis**: siehe Seite 13 für Erläuterung]

Die Legende von "Run & Boom"

Seit Jahrhunderten leben die beiden großen Elementarclans – der Wasser-Clan (Blau) und der Feuer-Clan (Rot) – in einer zerbrechlichen Balance. Beide Clans besitzen ungeheure Macht, doch keiner von beiden kontrolliert den Pfad der Elemente, eine mächtige Energiequelle.

Einmal pro Generation materialisiert sie sich tief im Niemandsland zwischen den beiden Reichen, hoch über den Wolken.

Dieses gelblich schimmernde Tor (der Checkpoint) aus reiner Energie ist der Schlüssel zur absoluten Dominanz über die Elemente. Damit der Kampf darum die Welt nicht zerstört, wurde ein uraltes Ritual geschaffen. Der Konflikt wird nicht auf der Erde ausgetragen, sondern in einer Umgebung, die beiden Elementen fremd ist: der Himmels-Arena. Hier, im Reich der Luft, sind Feuer und Wasser gleichsam verletzlich.

Das Ritual schreibt einen stetigen Wechsel vor: Nur ein Kämpfer darf den Pfad beschreiten (Runner) – und nur einer darf jagen (Kanone). Mit jeder Runde tauschen sich die Gewichte der Macht. Die Clans entsenden ihre besten Champions: „Blue“ vertritt den fließenden Geist des Wassers und „Red“ den unaufhaltsamen Zorn des Feuers.

Der Runner sprintet über schwebende Inseln durch die gefährliche Arena. Er muss blitzschnell die Ebenen wechseln, Hindernissen vorausschauend ausweichen und dem Chaos entgehen, um das Ziel zu erreichen.

Der Kanonen-Krieger gleitet als Jäger entlang der Ränder dieser Dimension. Er feuert explosive Elementarkugeln auf den Runner, um ihn mit aller Macht davon abzuhalten, die Energie für seinen Clan zu beanspruchen.

Es ist ein Tanz aus Präzision und Zerstörung – bis einer der Champions das gelbe Energie-Tor durchquert und das Schicksal der Elemente neu schreibt.

Inspiration:

Die Spielidee wurde nicht von bestehenden Spielen inspiriert, sondern ist viel mehr auf Grundlage von Ansprüchen an die Entwicklung und das Konzept entstanden:

- Kompetitiver Wettkampf: Unsere Intention bestand daraus, ein simples Spielkonzept wettkampforientiert umzusetzen.

- Entscheidend war für uns eine sehr simpel zu verstehende Mechanik, die aber trotzdem viel Raum zum Verbessern und Perfektionieren lässt, sodass man sich mit mehr Erfahrung immer weiter steigert. Dieser Faktor wird dadurch verstärkt, dass man auch an der Kanone taktisch vorgehen kann und seine Projektile clever zum richtigen Zeitpunkt abschießt.

- Konzept des Rollenwechsels: Jeder Spieler übernimmt im Laufe des Spiels abwechselnd beide Rollen und muss dementsprechend auch in beiden gute Fähigkeiten haben, um seinen Gegner besiegen zu können.

- Das Spiel wird von beiden Spielern lokal an einem gemeinsamen Eingabegerät gesteuert. Es ermöglicht hitzige emotionale Duelle.
  Das Ziel bestand darin, ein Spielkonzept ohne große Komplexität zu entwerfen, welches beispielsweise unter Freunden in Pausen an der Uni gespielt werden kann. Dennoch wollten wir den Lernfaktor nicht außen vor lassen: Nach zahlreichen gespielten Runden wird man die Muster der Plattformen besser einschätzen (aber nicht wiedererkennen) und tendenziell klügere taktische Entscheidungen treffen können als nach einer einzigen gespielten Runde.

- Die Idee der freien Bewegung innerhalb eines automatisch scrollenden Spielfelds wurde von den Auto-Scrolling-Leveln aus Super Mario Bros 3 inspiriert.

- Des Weiteren haben wir die Ursprüngliche Idee, dass der Runner springen und auf den Plattformen läuft verworfen und uns dafür entschieden, dass der mit einem Jetpack fliegt, hier haben wir uns von JetPack Joyride inspirieren lassen.

Designkonzept [**2) KI-Hinweis**: siehe Seite 13 für Erläuterung]

Beziehungen zwischen den Klassen und zentrale Methoden

In dem Spiel „Run & Boom“ stehen folgende Klassen miteinander in Beziehung. Zudem sind die jeweiligen zentralen Methoden innerhalb der Tabelle aufgeführt.

Klasse / Modul	Zweck (kurz)	Beziehungen (Vererbung: Komposition, Aggregation)	Zentrale Attribute (Auswahl)	Zentrale Methoden (Auswahl)
main.py	Einstiegspunkt: erstellt das Spiel und startet den Game-Loop.	• Abhängigkeit: nutzt Game		• main() => Game() + run()
Game (game.py)	Steuert Initialisierung, Zustände, UI und den Game-Loop (Context / Controller).	• Komposition: besitzt GameWorld, 2x Player • Komposition: verwaltet Sprite-Gruppen (all_sprites, obstacles, projectiles) • Aggregation: wird von Runner /Cannon/Projectile /Checkpoint als game-Referenz genutzt	• game_state, running • player1, player2 • current_runner current_cannon • all_sprites, obstacles, projectiles • current_round_num , last_point_reason	• run(), events(), update(dt), draw() • start_game(), start_round(), next_round() • switch_roles(), reset_game() • checkpoint_reached, cannon_scores(), process_round_result()
Game-World (gameworld .py)	Erstellt und aktualisiert die Spielwelt pro Runde (Spawning, Kollisionen, Rendering).	• Komposition: erstellt Runner, Cannon, Checkpoint je Runde • Abhängigkeit: nutzt ObstacleFactory (Factory Pattern) • Aggregation: hält Referenz auf Game + dessen Gruppen	• runner, cannon, checkpoint • scroll_speed • obstacle_spawn_interval, obstacle_spawn_timer	• setup_round(...) • update(dt), draw(screen) • spawn_obstacle(dt), spawn_initial_obstacle • is_lane_free(...), check_collisions()
Player (player.py)	Menschlicher Spieler: Name, Controls, Farbe, Rolle und Punkte.	• Komposition: wird von Game gehalten (2 Instanzen) • Abhängigkeit: controls werden aus settings übergeben	• name, color, controls • role • round_score, set_score, score	• win_round(), win_set() • switch_role() • reset()

Runner (runner.py)	Spielfigur links: Bewegung (x + Lanes), Kollisionen, kann aus dem Bild gedrückt werden.	• Vererbung: pg.sprite.Sprite • Aggregation: hält game-Referenz (Scoring/Sounds/obstacles) • Komposition: registriert sich in game.all_sprites	• pos, vel, acc • current_lane, target_lane • controls, color, key_states	• update(dt), get_keys() • is_target_lane_safe(lane) • collide_with_obstacle (obstacle) • reset_position(x, lane)
Cannon (cannon.py)	Kanone rechts: Lane-Bewegung und Schießen mit Cooldown.	• Vererbung: pg.sprite.Sprite • Aggregation: hält game-Referenz • Komposition: erzeugt Projektile beim Schuss	• pos, rect • current_lane, target_lane • shoot_cooldown, _left_was_pressed • controls, color	• update(dt), get_keys() • shoot() • get_lane_y(lane) • reset_position(start_lane)
Projectile (projectile.py)	Projektil: fliegt nach links, reagiert auf Treffer und wird entfernt.	• Vererbung: pg.sprite.Sprite • Aggregation: hält game-Referenz • Komposition: in game.all_sprites und game.projectiles	• pos, vel, speed • active • color, rect	• update(dt) • check_collision_with_runner(runner) • check_collision_with_obstacle(obstacle) • kill_me()/deactivate()
Obstacle & Obstacle Factory (obstacle.py)	Hindernisse scrollen nach links; Factory wählt Typ (short/long).	• Vererbung: Obstacle ist pg.sprite.Sprite • Factory Pattern: ObstacleFactory.create(...) • Komposition: Obstacle in game.all_sprites und game.obstacles	• obstacle_type • rect, image	• Obstacle.update(dt) • ObstacleFactory.create(game, x, lane)
Checkpoint (checkpoint.py)	Zielpunkt: erkennt Erreichen und meldet es an Game.	• Vererbung: pg.sprite.Sprite • Aggregation: hält game-Referenz • Callback: ruft game.checkpoint_reached() auf	• is_reached • rect, image	• update(dt) • check_reached(runner)
settings.py	Zentrale Konstanten (Balancing, Größen, UI, Steuerung).	• Abhängigkeit: wird von fast allen Modulen importiert	• Fenster: WIDTH /HEIGHT/FPS • Lanes: NUM_LANES, LANE_HEIGHT • Speed: SCROLL_SPEED, PROJEKTILE_SPEED • Controls/UI: PLAYER1/2_CONTROLS, UI_*	(nur Konstanten)

Refactoring und Modularisierung

Im Rahmen unseres Projekts „Run & Boom“ stellte das Refactoring des Quellcodes einen entscheidenden Entwicklungsfortschritt dar, um die anfänglich entwickelte Code-Struktur an die Anforderungen der gemeinsamen Teamarbeit im GitHub-Repository anzupassen.

In den Anfängen des Projekts entstanden durch erstes Ausprobieren, Testen und den Bau einfacher Prototypen mit der Python-Bibliothek „Pygame“ monolithische Code-Architekturen (eine Struktur, bei der alle Klassen, Funktionen und Komponenten des Codes in einer einzigen Datei vereint sind), die jeweils lokal auf Geräten einzelner Gruppenmitglieder gespeichert wurden.

Diese Strukturen, die oft als „Spaghetti-Code“ bezeichnet werden, führten im Laufe des Projekts, mit steigendem Funktionsumfang, schnell zu Unübersichtlichkeit und Verwirrung. Ebenso erschwerten die voneinander abweichenden lokalen Entwicklungsstände massiv die gemeinsame Zusammenarbeit im Team.

Um diesen Problemen entgegenzuwirken, nahmen wir unsere Arbeit im GitHub-Repository „PROG_2_RUN_AND_BOOM_TEAM04“ auf. Im Zuge dessen machten wir uns auch Gedanken über Refactoring und begannen, das „Single-Responsibility-Prinzip“ anzuwenden.

Zuvor hatte eine Zusammenführung der voneinander abweichenden lokalen Entwicklungsstände in einem Teammeeting stattgefunden, sodass ab diesem Zeitpunkt die Modularisierung des Codes auf Grundlage der fusionierten Version im GitHub-Repository starten konnte: Wir lagerten seit Beginn unserer Zusammenarbeit im GitHub-Repository einzelne Klassen in dedizierte Dateien innerhalb des Programmordners aus.

Dadurch verhalten sich die ausgelagerten Dateien wie Module und die Verbindung untereinander kann mittels gezielter Importe erfolgen.

So kümmert sich beispielsweise die ausgelagerte Datei „obstacle.py“ ausschließlich um die Hindernis-Logik und das Verhalten dieser im Spiel, ohne Abhängigkeiten zum Hauptmenü oder anderen spielunabhängigen Komponenten zu besitzen.

Ebenso fungiert „settings.py“ als zentraler Konfigurationsspeicher, dessen definierte Parameter und Variablen über die Anweisung „from settings import *“ anderen Klassen effizient zur Verfügung gestellt werden.

Dieses Vorgehen und die Methodik der Entkopplung brachten starke Vorteile hinsichtlich Übersichtlichkeit, Fehlersuche und Verständnis des Codes mit sich. Besonders ermöglichte es uns effiziente Teamarbeit anhand eines gemeinsamen Entwicklungsstandes im GitHub-Repository. Durch zahlreiche Commits von verschiedenen Gruppenmitgliedern wurde somit im GitHub-Repository eine finale Version des Spiels „Run & Boom“ entwickelt.

Reflektion

Im Rahmen unseres Projekts sind folgende generative KI-Tools für verschiedene Anwendungsfälle, zum Einsatz gekommen:

- Claude Sonnet - 4.5 (Anthropic)
- ChatGPT - 5 (OpenAI)
- Gemini 3 Pro (Google)

Im Rahmen dieses Projekts wurde hauptsächlich auf das KI-Modell von Anthropic „Claude Sonnet“ zugegriffen. Da mehrere Gruppenmitglieder zusätzlich ein ChatGPT Pro-Abonnement besitzen, wurde das Model GPT 5.0 (OpenAI) ebenfalls verwendet. Nach kurzer Zeit stellte sich jedoch heraus, dass Claude Sonnet bedeutend bessere Ergebnisse hinsichtlich Code-Verständnis, Code-Verarbeitung und Code-Generierung im Vergleich zu anderen Modellen lieferte. Somit wandten wir uns mit Problemen im Code, nachdem wir selbst nicht weiterkamen, primär an den Chatbot von Anthropic. Die Aufgaben reichten dabei von komplexer Fehlersuche im Code über Code-Generierung bis hin zu einfachem Feedback des aktuellen Entwicklungsstands.

Im Gegensatz dazu kamen die Modelle ChatGPT (OpenAI) und Gemini (Google) vorzugsweise bei Verständnisfragen und der Ausarbeitung von Ideen und Konzepten zum Einsatz.

Der Lerneffekt ist unserer Erfahrung nach am geringsten, wenn die gesamte Fehlersuche und das Bug-Fixing von einer generativen KI übernommen wird.

Die KI (vor allem Claude) ist in der Lage, auf Basis der Fehlermeldung und dem betroffenen Code in einer sehr hohen Geschwindigkeit, mit absoluter Genauigkeit und Effizienz die Ursache zu finden und Code zu korrigieren. Im Zuge dessen bleiben wichtige Verständnis- und Debugging-Skills auf der Strecke.
Bei einer Code-Generierung (z.B. einer zu erstellenden Methode einer bestimmten Klasse) mittels eines Chat-Bots verhält es sich unserer Erfahrung nach gegenteilig. Dadurch, dass beim Prompt und Input der KI aktiv darüber nachgedacht werden muss, wie man das jeweilige Problem lösen könnte, ist man gedanklich schon in ein tieferes Verständnis (z.B. der Methode) abgetaucht: Welche Bedingungen müssen erfüllt werden? Welche Funktionen soll das Ergebnis haben? An welcher Stelle im Code soll das Ergebnis eingepflegt werden? usw. Durch diese Gedankengänge im Voraus konnte bei dieser Nutzung ein deutlich größerer Lerneffekt erzielt werden.

In dem Fall der Entwicklung von „Run & Boom“ war es daher sehr hilfreich, dass jedes Gruppenmitglied eine einheitliche Vision und Vorstellung des fertigen Spiels im Kopf hatte. Dieses Verständnis spiegelte sich in detaillierten und effizienten Prompts wieder, wobei im Zuge dessen der generierte Output viel besser nachvollzogen werden konnte.

Darüber hinaus legten wir im Team, am Anfang des Projekts, die klare Regel fest, keinen KI generierten Code stumpf zu kopieren, sondern die Ergebnisse von Chatbots abzutippen und ausführlich zu kommentieren, wenn wir sie im Code integrieren.

Ebenso wurde die Code-Qualität positiv durch die KI-Nutzung beeinflusst. Es wurde in regelmäßigen Zeitintervallen Feedback zur Struktur- und Code-Qualität von Claude generiert, worauf gegebenenfalls Anpassungen und Umdenken folgten.

Zusammenfassend lässt sich sagen, dass die KI-Nutzung den Gruppenmitgliedern eine große Hilfe bezüglich sauberer Struktur, Feedback und Code-Qualität war.

Im Rahmen des Feedbacks von KI-Modellen zum Code musste beim Prompt oftmals Überzeugungsarbeit geleistet werden, damit positives, besonders aber negatives Feedback und Verbesserungsvorschläge als Output zurückkamen. Beispielsweise musste „hartes und ehrliches Feedback aus der Sicht eines Weltklasse-Programmierers mit 30 Jahren Berufserfahrung“ gefordert werden. Andernfalls lieferten alle genutzten Chatmodelle durchgehend positive Rückmeldung und es wurde kaum bedacht, negative Kritikpunkte zu äußern.
Kritisch betrachtet könnten die im Rahmen dieses Projekts verwendeten KI-Tools den Entwicklern einen Großteil der Arbeit abnehmen, sie jedoch nicht vollständig ersetzen.

Da durch KI-Nutzung, vor allem in Lernprozessen, die mentale Verarbeitung sinkt, die Planungsfähigkeit ersetzt wird und die Konzentrations- und Gedächtnisleistung teilweise an die KI ausgelagert wird, stellt dies einen Balanceakt in der heutigen Zeit dar.
Im Gegensatz dazu würde man durch ein Zugriffsverbot und Verzicht auf solche Systeme enorme Unterstützung und Effizienz bei ausgewählten Aufgaben verschenken.
Daher wurde im Zuge der Durchführung dieses Projekts versucht, die Lernziele gleichermaßen mit eigenem Verständnis zu erreichen, ebenso aber auf den effizienten Umgang mit KI-Tools bei spezifischen Aufgaben nicht zu verzichten.

Abgesehen von der Reflektion über KI-Nutzung wurden in der finalen Version von „Run & Boom“ geplante Eigenschaften, Spielelemente und Systeme teils anders oder gar nicht umgesetzt als ursprünglich geplant.
So wurde sich bewusst dagegen entschieden Power-Ups, wie zum Beispiel Schutzschilder für den Runner vor Projektilen zu implementieren. Diese Entscheidung wurde im Team mit der Begründung getroffen, das Spiel simpel zu halten und nicht mit Features zu überlagern. Anstelle von Power-Ups wurde Wert auf faire Einstellung von spielentscheidenden Parametern wie den Scroll-Speed, den Switch-Lane-Speed oder die Geschwindigkeit von Projektilen gelegt.

Aus denselben Gründen wurde die Implementierung von halben Hindernissen (Grasblöcken), bei denen sich der Runner hätte ducken müssen, vernachlässigt.

Weiterentwickelt wurde jedoch das Punktesystem: Es wurde ein Runden- und Satzsystem eingeführt, welches durch Gewinnsätze für zusätzliche Anspannung und Konzentration bei den Spielern sorgt.

Darüber hinaus wurde die Kanone und der Runner mit einem Jetpack ausgestattet und das Duell der Spieler in die Luft verlegt. Der Spieltitel „Run & Boom“ ist jedoch gleichgeblieben, wobei der Begriff „Fly & Boom“ zur veränderten Variante womöglich besser passen würde. Das Team wollte ihren anfänglichen Titel aber bewusst beibehalten.

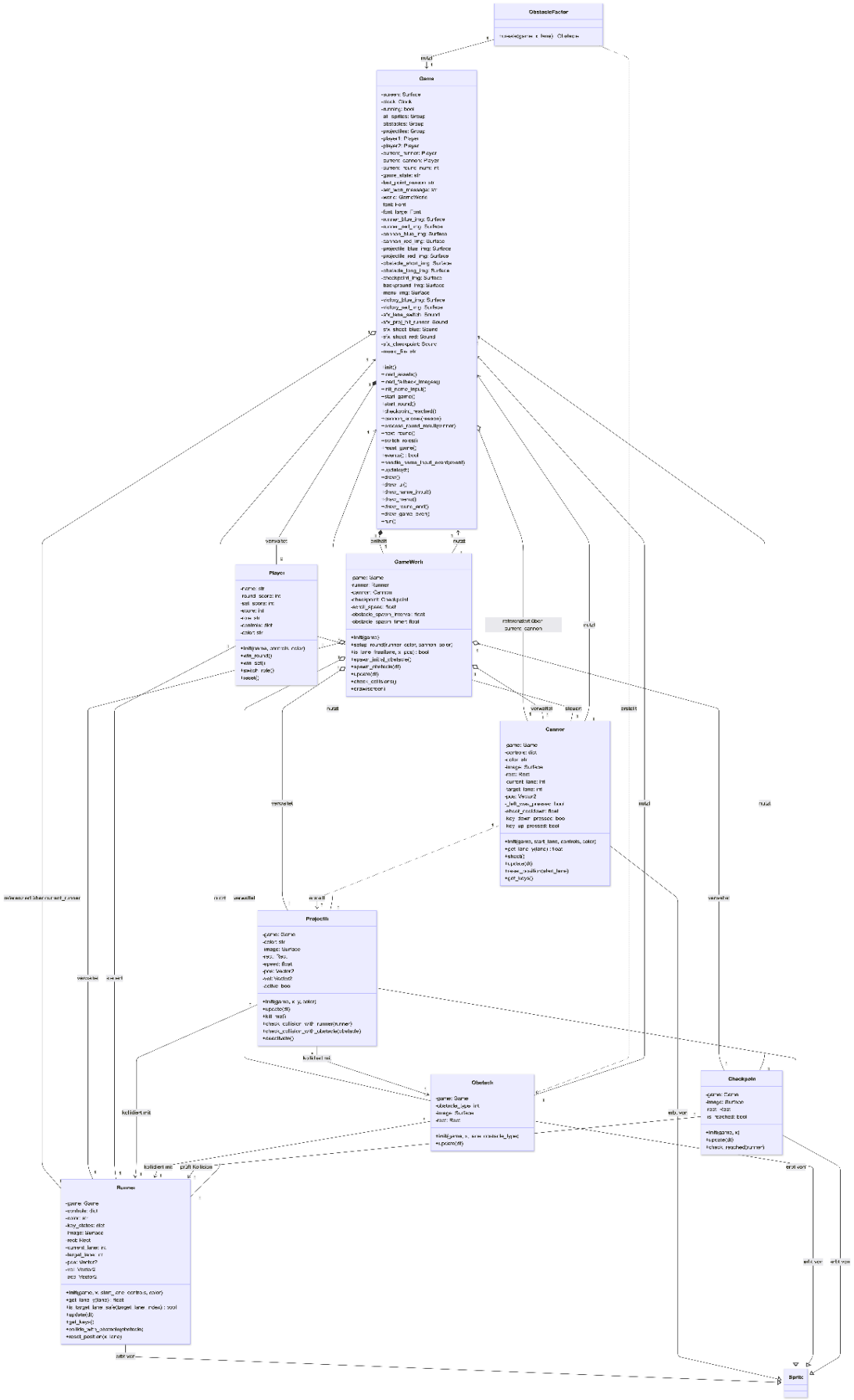
Arbeitsmatrix

Im Folgenden wird die Arbeitsaufteilung des Teams 04 für die Entwicklung von „Run & Boom“ anhand einer Arbeitsmatrix dargestellt. Dabei wird der Arbeitsaufwand von 100% auf die jeweiligen Gruppenmitglieder gleichermaßen aufgeteilt. Da alle Gruppenmitglieder gleichermaßen an der Code-Erstellung und dem Testen mitgewirkt haben, sind darüber hinaus Spezialisierungen aufgelistet.

Arbeitsmatrix Team 04 – „Run & Boom“			
100 % des Arbeitsaufwandes			
Niclas Scharlach	Tim-Ole Pries	Jonte Lahrs	Mares Büchler
25 %	25%	25%	25%
Spezialisierung: Startbildschirm, Spielmenü	Spezialisierung: Spielgrafiken & Sounds	Spezialisierung: Projektleitung, Grafiken & Sounds	Spezialisierung: Abschluss-Dokumentation

UML-Diagramm des Spiels „Run & Boom“ (finale Version)

[**3) KI-Hinweis**: siehe Seite 14 für Erläuterung]



Hinweise zur Nutzung von generativer KI in der Abschlussdokumentation

1) Spielkonzept: Story hinter „Run & Boom“ (siehe Seite 4)

Genutztes Modell: Gemini 3 Pro (Google) (im Think-Modus)

Use-Case: Es wurde KI-Unterstützung beim Entwickeln und Ausformulieren der Story hinter „Rund and Boom“ in Anspruch genommen. Jedoch wurden die wichtigsten Rahmenbedingungen, Namen und der Gedanke hinter der Story mit in den Prompt eingepflegt.

Dazugehöriger Prompt: „Schreibe eine epische Hintergrundgeschichte mit dem Titel 'Die Legende von Run & Boom' die den ewigen Konflikt zwischen dem Wasser-Clan (Blau) und dem Feuer-Clan (Rot) thematisiert. Verlege den Schauplatz in eine neutrale Himmels-Arena, da Luft das einzige Element ist in dem beide Parteien verwundbar sind. Beschreibe die Spielmechanik als ein uraltes Ritual des stetigen Rollenwechsels bei dem ein 'Runner' versucht ein gelbes Energietor (den Checkpoint) zu erreichen, während er vom gegnerischen 'Cannon-Krieger' gejagt wird.“

2) Designkonzept: Vergleich der Klassenstruktur: Entwurf vs. finaler Code (siehe Seite 6)

Genutztes Modell: ChatGPT-5.2 (OpenAI)

Use-Case: Es wurde mit Hilfe einer KI (GPT-5.2) eine Tabelle entworfen, welche sowohl die Beziehung zwischen den Klassen als auch die Methoden und Attribute erläutert.

Dazugehöriger Prompt: „Aufgabe: "Beziehungen zwischen den Klassen: Erläutern Sie die Beziehungen (z.B. Vererbung, Komposition, Aggregation) zwischen Ihren Klassen. Methoden und Attribute: Beschreiben Sie zentrale Methoden und Attribute und erläutern Sie deren Aufgabe kurz und verständlich." Löse diese Aufgabe, sodass ich diesen Teil ergänzen kann in der Dokumentation des Spiel. Wichtig ist dass es eine Tabelle sein soll. Des Weiteren soll diese einfach und übersichtlich sein ohne viele oder lange Erläuterung. eine Erläuterung oder Erklärung soll n Stichpunkten oder in ganz prägnanten kurzen Sätzen sein. “ Des Weiteren wurde der Code als Python Quelldatei dem Chat hinzugefügt.

3) Erstellen des UML-Klassendiagramms: finaler Code (siehe Seite 12)

Genutztes Modell: Claude Sonnet 4.5 (Anthropic)

Use Case: Der Quellcode des verwendeten UML-Diagramms wurde mittels Claude Sonnet auf Grundlage des finalen Codes des Spiels „Run & Boom“ generiert.

Anschließend wurde der „Markdown Mermaid“ Code (siehe Textdatei im Anhang) mit der Website: https://www.mermaidchart.com/ visualisiert.

Dazugehöriger Prompt:

„Ich möchte aus diesem finalen Code des Spiels „Run & Boom“ ein UML-Diagramm entwickeln. Dabei möchte ich einen Code erhalten, der in der VS-Code Erweiterung "Markdown Preview Mermaid Support" als md.-Datei ausführbar ist. Schreibe mir einen dementsprechenden Code, der die gesamten Beziehungen der Klassen, Methoden und Attribute vollständig in einem UML-Diagramm abdeckt.

Hinweis: Die finale Umsetzung in der Preview wurde dennoch final über die angegebene Website und nicht in der VS-Code Erweiterung visualisiert.

Inhalt des Anhangs:

- Allgemeine Informationen der Gruppenmitglieder
- Zip-Datei: Programmordner „Run & Boom“
- Quellcode des UML-Diagramms als Textdatei (TXT)
- Zusätzlich: UML-Diagramm als Foto (PNG)
- Zusätzlich: Arbeitsmatrix als Foto (PNG)

Allgemeine Informationen der Gruppenmitglieder

Gruppe: Team 04

Gruppenmitglieder:

- Niclas Scharlach (947394)
- Tim-Ole Pries (947387)
- Jonte Lahrs (947064)
- Mares Büchler (946607)