

Screenshot ingame Battle-Pong

Battle-Pong

**Zwei Spieler Pong mit, an OpenAI Gym angelehnter, API zum
Trainieren von Reinforcement Algorithmen**

Frederic Aust

Zuletzt bearbeitet am 9. April 2021

Hausarbeit im Rahmen des Moduls
Vertiefung Simulation
von Professor Frochte

Vielen Dank an die Masterstudenten
Christian Janorschke und Stefan Simanek,
für Ihre konstruktive Kritik und guten Ideen während der
Entwicklung dieses Projektes.

Inhaltsverzeichnis

1	Abstract	3
2	Einleitung	4
3	Grundlagen	5
3.1	Pong	5
3.2	OpenAI Gym	6
3.2.1	observation	6
3.2.2	reward	6
3.2.3	done	6
3.2.4	info	6
3.3	Godot	7
3.4	Websocket	7
3.5	JSON	8
3.6	Base64	8
3.7	Herausforderungen	9
3.7.1	Bestimmung des Abprallwinkels	9
3.7.2	Spielzeit entkoppeln	9
3.7.3	Screenshot der Oberfläche übertragen	9
4	Planung, Entwicklung und Nutzung	10
4.1	Battle-Pong GUI	10
4.1.1	Main menu	10
4.1.1.1	Two Local Player	11
4.1.1.2	Learn with Images	11
4.1.1.3	Learn with Position	12
4.1.1.4	Training	12
4.1.2	Settings-Window	12
4.1.3	Settingsdatei	15
4.1.4	Anpassung der Tastatursteuerung	16
4.1.5	Headless	18
4.2	Battle-Pong Code und Algorithmen	19
4.2.1	Ball Physik	19
4.2.1.1	Startwinkel	20
4.2.1.2	Abprallwinkel an Spieler	20
4.2.2	Spieler Physik	21
4.2.3	Obstacle Physik	21
4.2.4	Entkopplung der Zeit	22

4.2.5	Bewegungsreihenfolge	22
4.2.6	Aufnahme und Übertragung der Screenshots	23
4.2.7	”yield” - warum die Wartefunktionen notwendig sind	23
4.3	BattlePongAI	24
4.3.1	API	24
4.3.1.1	Instanziierung von Gym	24
4.3.1.2	connect_player()	24
4.3.1.3	reset()	25
4.3.1.4	step(action)	25
4.3.1.5	Aufbau Observation	25
4.3.2	Beispiel Agent RoboCat	27
4.4	Kommunikation	27
4.4.1	BattlePongAI \longrightarrow Battle-Pong	28
4.4.2	BattlePongAI \longleftarrow Battle-Pong	28
4.5	Installationsanleitung	28
4.5.1	Godot	28
4.5.2	Python	29
4.5.3	Test	29
5	Ergebnisse	30
5.1	Godot	30
5.2	Python	30
6	Fazit (und Ausblick)	31

1 Abstract

The idea of this project is to create a two player version of the classic game Pong where the players will be controlled by reinforcement algorithms (agents).

It is developed with the game engine Godot and the communication API is forked from Gym, a toolkit created from OpenAI.

Agents can learn to play the game while observing the differences between images or with information about position and ball velocity. The data will be transmitted via Websocket, which allow a duplex communication over TCP. Agents can be implemented in python and a default implementation with a nonlearning agent is provided. All in all is it a simple game engine and using the websocket protocol a good choice for this project. It is quite simple to create an agent to control the players.

2 Einleitung

In diesem Projekt für das Modul Vertiefung Simulation geht es um die Planung und Entwicklung des Spieleklassikers Pong als Zwei-Spieler Version. Eine Singleplayer Variante von Pong ist im Gym toolkit implementiert und dient als Vorlage. Die Herausforderung besteht darin die genutzte API zu verwenden und so umzubauen, dass aus dem Singleplayer ein CO-OP Spiel wird. Das bedeutet, dass anstelle eines Agenten zur Steuerung des Spielers zwei Agenten verwendet werden, welche jeweils einen Spieler steuern und so gegeneinander spielen können.

Daraus ergibt sich der Projektname "Battle-Pong".

Die steuernden Agenten sollen in Python implementiert werden. Wie diese mit dem Spiel kommunizieren ist nicht festgelegt und wird im Projektverlauf recherchiert, geplant und getestet.

Zudem sollen zwei Lernmodi zur Verfügung stehen:

1. Lernen über die Positionen und den Bewegungsvektor des Balls (Q-learning)
2. Lernen über Bilder (double Q-learning)

Als weitere Herausforderung an die Agenten kann ein bewegliches Hindernis (Obstacle) eingeschaltet werden. Dieses bewegt sich mittig von oben nach unten.

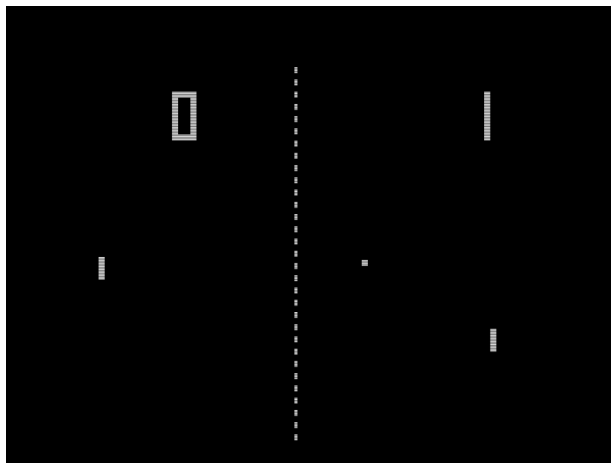
3 Grundlagen

In diesem Kapitel werden die, für die weitere Hausarbeit notwendigen, Grundlagen erklärt. Zusätzlich wird auf vertiefende Quellen verwiesen. Vorausgesetzt wird Grundlagenwissen über die Programmiersprache Python, sowie die Bedeutung von Agenten im Zusammenhang mit Maschinellen Lernen. Ein sicherer Umgang mit Vektorrechnung ist von Vorteil.

3.1 Pong

Das Spiel Pong wurde im Jahr 1971 von Nolan Bushnell entwickelt und beruht auf dem Spiel Tischtennis (Ping Pong), woraus sich auch der Name "Pong" ableitet. Zu Beginn wurde das Spiel nicht, wie heute auf einem Computer, programmiert sondern in Platinen "gegossen".

Das 2D-Spiel besteht aus zwei vertikalen, kurzen Linien (Spieler) auf gegenüberliegenden Seiten und einem sich dazwischen bewegendem Quadrat. Die Aufgabe der Spieler ist es, ähnlich wie beim Tischtennis, einen Ball daran zu hindern auf der eigenen Seite das Spielfeld zu verlassen. Die Spieler können sich nur hoch und runter bewegen, um den Ball abzuwehren. Am oberen und unteren Rand prallt der Ball ab, ebenso an den Spielern. Ziel des Spiels ist es den Ball am gegnerischen Spieler vorbei aus dem Spielfeld zu befördern. [Interview N. Bushnell]



Quelle: <https://de.wikipedia.org/wiki/Pong#/media/Datei:Pong.svg>, Wikipedia

Abbildung 3.1: Screenshot des Spiels Pong von Atari, veröffentlicht 1972

3.2 OpenAI Gym

Gym ist ein Toolkit von OpenAI, welches das Erlernen und Arbeiten mit Reinforcement Algorithmen vereinfachen soll. Zudem wird hier eine einheitliche API bereitgestellt mit der es möglich ist verschiedene Algorithmen miteinander zu vergleichen. Des Weiteren werden bereits einige Spiele bereitgestellt, mit denen gearbeitet werden kann.

Der Informationsaustausch zwischen Agenten und Spiel besteht aus einem **Step** (definierte Aktion aus einer Liste möglicher Aktionen). Dieser teilt dem Spiel mit welche Aktion im nächsten Zeitfenster durchgeführt werden soll. Anschließend werden Information über die Spielumgebung zurückgegeben. Diese Rückmeldung besteht aus den vier Punkten **observation**, **reward**, **done** und **info**.

Mit den Funktionen *make* und *close* kann die Umgebung erzeugt bzw. geschlossen werden. Durch Aufruf von *reset()* wird die Umgebung in einen Ursprungszustand zurückgesetzt.

3.2.1 observation

In der Observation stehen alle für den Agenten notwendigen Informationen um sich in der der Spielumgebung orientieren zu können.

Beispiel: Position und Ausrichtung des Spielers, Bewegungsgeschwindigkeit oder Sensorwerte

3.2.2 reward

Damit die Agenten aus Spielsituationen lernen können werden Rewards (Punkte) vergeben. Bei erwünschten Resultaten werden positive Rewards vergeben, während bei unerwünschten Resultaten negative vergeben werden. Damit die Agenten nicht versuchen möglichst lange in einer Situation zu bleiben, um negative Rewards zu vermeiden, kann es sinnvoll sein die Rewards an die Spielzeit oder Anzahl der Steps zu koppeln und so zu reduzieren. Die Vergabe der Rewards ist allerdings immer abhängig vom verwendeten Spiel und der zu lernenden Aufgabe.

Ziel ist es die Summe der Rewards zu maximieren.

3.2.3 done

Der Parameter Done gibt Auskunft darüber, ob das Spiel noch läuft oder ob es bereits beendet wurde.

Auch kann darüber erkannt werden, dass die Umgebung zurückgesetzt werden muss.

3.2.4 info

Über den vierten Parameter können zusätzliche Informationen ausgegeben werden, welche beim Debuggen hilfreich sein können. Der Aufbau ist nicht festgelegt, da diese Werte offiziell nicht vom Agenten zum Lernen verwendet werden dürfen.

3.3 Godot

Godot ist eine Spiele-Engine die kostenfrei und Quell-Offen unter MIT Lizenz steht.[Godot]
Im Gegensatz zu den proprietären Spiele-Engines wie Unity und der Unreal Engine gibt es keine Limitierung bei der Nutzung und Vermarktung. Godot erhebt keine Ansprüche an die erstellten Spiele. Es ist kein Kostenmodell hinterlegt, das zwischen Hobby-, Indie-, und Professionellen Spieleentwicklern unterscheidet.

Die Spiele können für die gängigen Betriebssysteme exportiert werden, was die Entwicklung für verschiedene Plattformen vereinfacht. Zudem steht die IDE für die gängigen Betriebssysteme bereit. Die Entwicklungsumgebung, in der die Spiele erstellt werden, basiert selbst auf Godot.

Als Programmiersprachen[Godot Programmiersprachen] werden GDScript, VisualScript, C# und GDNative unterstützt. Für dieses Projekt wird GDScript verwendet, da es Python sehr ähnlich ist und daher die Einarbeitung vereinfacht.

Das Besondere an dieser Spiele-Engine ist der Aufbau durch Nodes. Die Nodes werden für sich definiert und können in einer Baumstruktur verschachtelt werden. So können aus einfachen Objekten komplexe Gebilde erstellt werden. Außerdem werden Sie über ihren Namen oder die Position im Baum angesprochen.

Für den Einstieg ist das Tutorial [Step-By-Step Anleitung] zu empfehlen, welche in einem zusammenfassenden Spiel [Your first Game] mündet.

Link zur Website:

<https://godotengine.org/>

3.4 Websocket

Ein, auf TCP aufbauendes, Netzwerkprotokoll bei dem Nachrichten zwischen Client und Server bi-direktional ausgetauscht werden.

Dabei öffnet der Client den Kommunikationskanal zu Server, welcher bestehen bleibt, bis die Antwort vom Server zurückgeschickt wird.

Eine gute, verständliche Lektüre zum Einlesen:

<https://www.heise.de/developer/artikel/WebSocket-Annaeherung-an-Echtzeit-im-Web-1260111.html>

Dokumentation zum Python Paket websockets:

<https://websockets.readthedocs.io/en/stable/>

Dokumentation zur Klasse WebSocketServer in Godot:

https://docs.godotengine.org/en/stable/classes/class_websocketserver.html

3.5 JSON

Das Datenübertragungsformat ist so gestaltet, dass es sowohl von Menschen, als auch von Programmen leicht zu verstehen / verarbeiten ist.

Dabei basiert es auf zwei Arten von Zuweisungen:

1. Variable: Wert
2. Geordnete Liste, wobei die Listenelemente sowohl Variablen, als auch weitere Listen sein können

Vertiefende Informationen über Aufbau und Verwendung: <https://www.json.org/json-de.html>

3.6 Base64

Mit Base64 [Base64 Encoding] werden 8-Bit Binärdaten in lesbare ASCII-Zeichen konvertiert. Diese gehören zur Standard-Zeichensatztafel und sind damit überall verfügbar. Vorteil dieser Konvertierung ist die einfache Übertragung, da dem Zahlenwert Buchstaben und Zeichen zugeordnet werden und reine 8-Bit Werte, beispielsweise auch für Steuerzeichen stehen können. Dies kann sonst bei der Übertragung zu Problemen führen.

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Quelle: <https://tools.ietf.org/html/rfc2045.html#section-6.8>, RFC 2045

Tabelle 3.1: The Base64 Alphabet

3.7 Herausforderungen

3.7.1 Bestimmung des Abprallwinkels

Es muss ein Algorithmus entwickelt werden, mit dem die Abprallwinkel bestimmt werden. Dabei soll der Ball von Spielern

3.7.2 Spielzeit entkoppeln

3.7.3 Screenshot der Oberfläche übertragen

4 Planung, Entwicklung und Nutzung

4.1 Battle-Pong GUI

Das Spiel Battle-Pong besteht aus drei Bereichen:

1. Main menu (Hauptmenü)
2. Game (Spiel)
3. Settings Window (Spieleinstellungen)

Bei Programmstart wird zuerst das Hauptmenü angezeigt. In diesem kann einer der verschiedenen Spielmodi ausgewählt oder die Spieleinstellungen (Settings Window) geöffnet werden.

Es ist auch möglich das Spiel direkt (ohne Hauptmenü) zu starten, was besonders für den automatisierten Einsatz beispielsweise auf Servern praktisch ist. Allerdings müssen hierfür ein paar Einstellungen im Spiel verändert werden, dazu im Kapitel Headless (siehe 4.1.5) mehr.

4.1.1 Main menu

Im Hauptmenü kann der Spielmodus gewählt oder das Fenster Settings aufgerufen werden, um Spieleinstellungen zu verändern. Zudem können Spielmodifikationen eingeschaltet werden, mit denen das Spielerlebnis verändert wird. Derzeit steht nur der Modifikator Obstacles zur Verfügung. Dadurch wird ein Block erzeugt, der sich in der Mitte des Spielfeldes von oben nach unten bewegt.

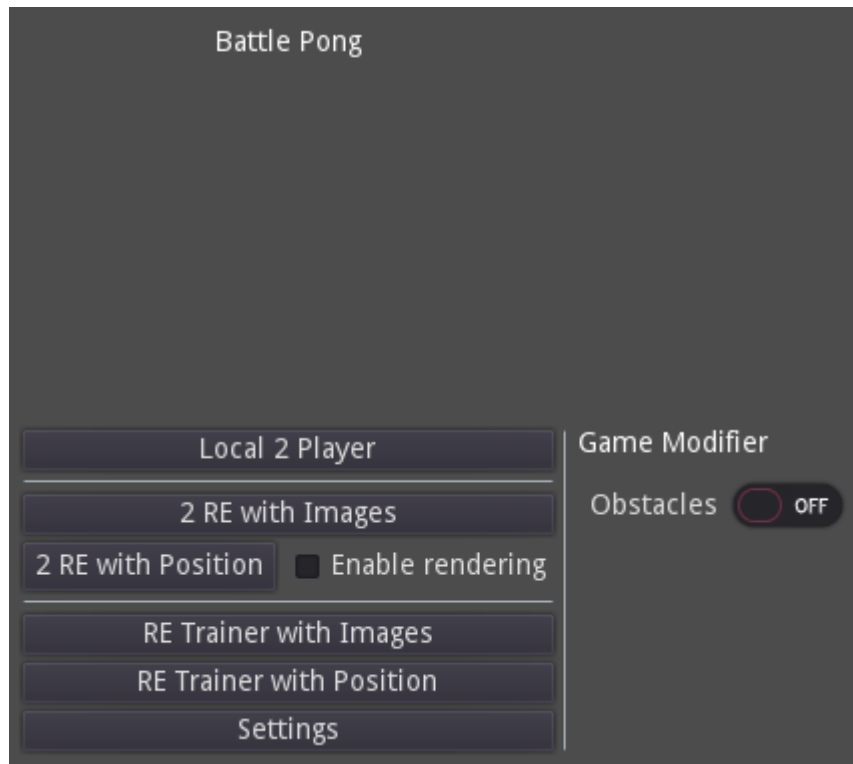


Abbildung 4.1: Battle-Pong Hauptmenü

4.1.1.1 Two Local Player

Damit man ein Gefühl für das Spiel entwickeln kann, werden im Modus *Two local Player* beide Spieler über die Tastatur gesteuert.

Für den Spieler 1 (links) werden die Tasten **W** und **S** verwenden.

Der Spieler 2 auf der rechten Seite wird mit den Pfeiltasten hoch **↑** und runter **↓** gesteuert. Für diesen Modus ist kein weiteres Programm notwendig.

4.1.1.2 Learn with Images

Soll dem Agenten ein Screenshot geschickt werden, ist es notwendig diesen Modus auszuwählen. Dabei wird der, nach Ablauf der Step-Time angezeigte, Bildschirminhalt aufgezeichnet, verkleinert und entweder in S/W oder in RGB8 konvertiert. Das konvertierte Bild wird dann, zusätzlich zu den Positionsangaben, an die Agenten übertragen.

Zusätzlich werden der Punktestand und andere Meldungen ausgeblendet. Dies ist der langsamste Modus, da das Rendern (insbesondere das Warten darauf, dass das Bild vollständig dargestellt wurde) einen spürbaren Effekt auf die Geschwindigkeit hat.

4.1.1.3 Learn with Position

In diesem Modus wird nur die Position der Spieler, des Balls und der Bewegungsvektor des Balls übertragen. Um die Geschwindigkeit zu erhöhen ist das Rendern standardmäßig ausgeschaltet. Es kann allerdings manuell im Hauptmenü eingeschaltet werden.

4.1.1.4 Training

Bei diesem Modus wird einer der Spieler (es können auch beide gleichzeitig trainiert werden) über die Tastatur gesteuert und die Eingabe während der Spielschritte zuzüglich Bild (Learn with Images) oder Positionsangaben (Learn with position) übertragen. Dadurch kann der aktuellen Situation eine "sinnvolle" Reaktion zugeordnet werden, welche im Gedächtnis des Agenten gespeichert wird.

4.1.2 Settings-Window

In dem Settings Window können allgemeine Einstellungen, Geschwindigkeit und Bildausgabe angepasst werden.

Game Settings	
Game:	
Playtime/Step (Seconds)	0.1
Wins till end	21
Port	9080
Image:	
Output RGB8	OFF
Height	60
Width	100
Trainer:	
Use realtime	ON
Ball:	
Speed Min	300
Speed Max	600
Speed Increment	50
Player 1:	
Speed	300
Player 2:	
Speed	300
Obstacle:	
Speed	500
Save	
Cancel	
Reset	

Abbildung 4.2: Battle-Pong Game Settings

- Game:
 - Playtime/Step (Seconds):
Dauer eines Spielschrittes.
Sinnvoll sind Werte zwischen 0,1 - 0,001 Sekunden.
 - Wins till end:
Siege die benötigt werden, bis das Spiel zurückgesetzt wird.
 - Port:
Portangabe, über den der Websocket-Server angesprochen werden kann
- Image: (Spielmodus Learn with Images)
 - Output RGB8:
Off => Bild wird in SW8 konvertiert (Graustufen)

- On => Bild wird in RGB8 konvertiert (farbig)
 - Details unter Godot Image Convert Funktion [Formate]
- Height:
 - Höhe des übertragenen Bildes in Pixel
- Width:
 - Breite des übertragenen Bildes in Pixel
- Trainer: (Spielmodus Trainer)
 - Use realtime:
 - ON => Es wird bei jedem Spielschritt die Anzahl an Sekunden gewartet, welche unter Playtime/Step eingestellt ist. Die tatsächliche Spielzeit entspricht der realen Zeit.
 - OFF => Sobald beide Spieler per Websocket ihre Befehle erhalten haben wird der Spielschritt ausgeführt und das Ergebnis zurückgesendet.
 - Achtung: Es kann je nach eingestellter Schrittzeit zu einer enormen Geschwindigkeitserhöhung führen, wenn realtime ausgeschaltet ist, da die Spielzeit von der ingame Zeit entkoppelt ist und die tatsächliche Dauer eines Spielschritts nur noch von der Kommunikation und internen Berechnungen abhängt.
- Ball:
 - Speed Min:
 - Geschwindigkeitsfaktor, mit dem der Ball zu Rundenbeginn beschleunigt wird
 - Speed Max:
 - Maximaler Geschwindigkeitsfaktor
 - Speed Increment:
 - Konstante, um die der aktuelle Geschwindigkeitsfaktor des Balls erhöht wird, sobald Dieser einen Spieler berührt.
- Player 1:
 - Speed:
 - Geschwindigkeitsfaktor von Spieler 1 (links)
- Player 2:
 - Speed:
 - Geschwindigkeitsfaktor von Spieler 2 (rechts)
- Obstacle:
 - Speed:
 - Geschwindigkeitsfaktor des Hindernisses
- Save:
 - Speichert die Einstellungen in data.json und wechselt zum Hauptmenü

- Cancel
Verwirft nicht gespeicherte Änderungen und geht zurück zum Hauptmenü
- Reset
Lädt die im Code hinterlegten Standardwerte, speichert Sie jedoch nicht

4.1.3 Settingsdatei

In der Settingsdatei sind die Spieleinstellungen hinterlegt. Es wurde das JSON (siehe 3.5) Format gewählt, da es innerhalb von Godot gut interpretiert und bearbeitet werden kann. Zudem ist es gut von Menschen lesbar, wodurch die Einstellungen gut debuggt werden können.

Des Weiteren kann die Datei **data.json** einfach ausgetauscht oder bearbeitet werden. Die Änderungen werden beim Neustart des Spiels geladen.

Die Namen der Variablen entsprechen den Feldern im **Settings Window** (siehe 4.1.2).

Listing 4.1: Generierte Settingsdatei: data.json

```
1 {
2     "ball": {
3         "height": 30,
4         "speed_increment": 50,
5         "speed_max": 600,
6         "speed_min": 300,
7         "width": 30
8     },
9     "game": {
10         "playtime_per_step": 0.1,
11         "port": 9080,
12         "wins_to_reset": 21
13     },
14     "image": {
15         "format": "L8",
16         "height": 60,
17         "width": 100
18     },
19     "player_one": {
20         "length": 60,
21         "speed": 300
22     },
23     "player_two": {
24         "length": 60,
25         "speed": 300
26     },
27     "trainer": {
```

```

28     "ip": "127.0.0.1",
29     "port": 9080,
30     "position": "Left",
31     "realtime_enabled": true
32 }
33 }

```

4.1.4 Anpassung der Tastatursteuerung

In Godot werden über die **Input Map** (Abb. 4.4) Tastenbezeichnungen zugewiesen. Dadurch wird einer Taste ein Alias zugewiesen, welcher die Lesbarkeit des Codes erhöht und zudem ermöglicht die Tastenbelegung nachträglich einfach zu ändern.

Um die Tastenbelegung anzupassen:

1. Innerhalb des Editors in der Menüleiste auf **Project** klicken und anschließend **Project Settings** (Abb. 4.3) auswählen.

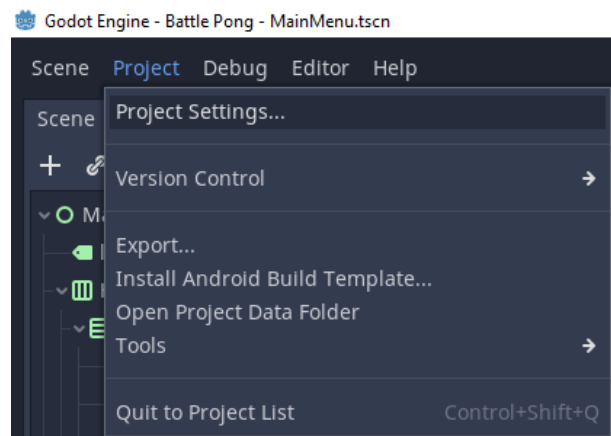


Abbildung 4.3: Godot - Menüleiste - Project

2. Anschließend den Reiter **Input Map** (Abb. 4.4) wählen.
Dort sind alle zu den Tastaturbefehlen gemappten Bezeichnungen aufgelistet.

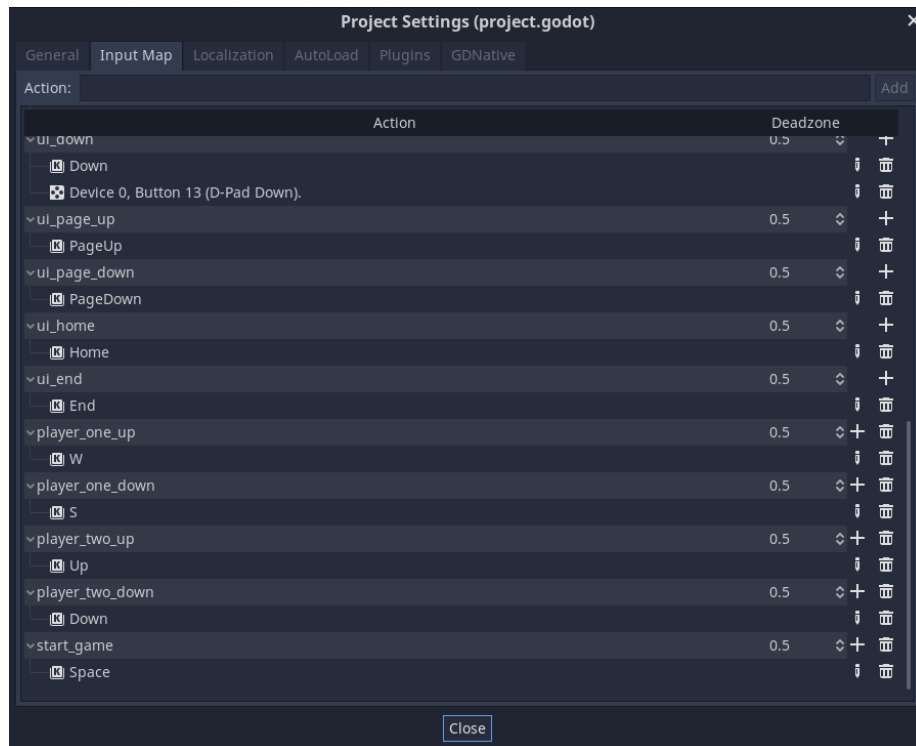


Abbildung 4.4: Godot - Project Settings - Input Map

- Um eine Zuweisung zu bearbeiten muss auf den kleinen Stift, auf der rechten Seite in der zu bearbeitenden Zeile, geklickt werden.
- Nun muss die Taste gedrückt werden, welche nun der Bezeichnung zugewiesen werden soll (Abb. 4.5).

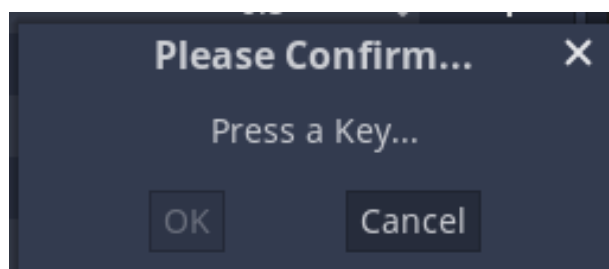


Abbildung 4.5: Godot - Settings - Edit Input Map

- Abschließend die Eingabe durch **OK** bestätigen

4.1.5 Headless

Ein richtiger Headless Modus ist derzeit noch nicht möglich.

Allerdings kann das Hauptmenü übersprungen und direkt der gewünschte Spielmodus gestartet werden.

Dafür müssen in der Godot Entwicklungsumgebung die Projekteinstellungen und in der Datei **GameSettings.gd** die Initialwerte einiger Variablen angepasst werden.

1. Innerhalb des Editors in der Menüzeile auf **Project** klicken und anschließend **Project Settings** (Abb. 4.6) auswählen.

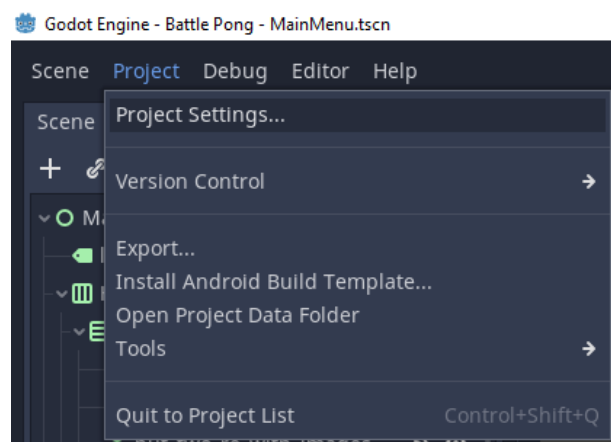


Abbildung 4.6: Godot - Menüzeile - Project

2. Danach den Reiter **General** wählen und dort **run** (Abb.s 4.7) öffnen

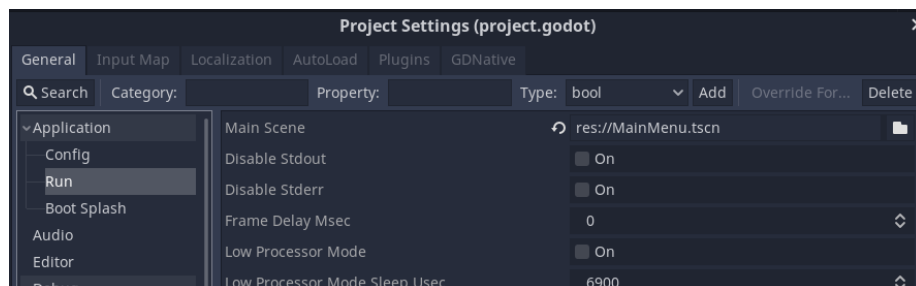


Abbildung 4.7: Godot - Project Settings - General - Run

3. Standardmäßig ist hier unter **Main Scene** die Datei *res://MainMenu.tscn* angegeben. Um direkt das Spiel zu starten muss die Datei *res://Main.tscn* ausgewählt sein
4. Die Änderung wird automatisch gespeichert

In der Datei `/root/GameSettings.gd` müssen die Standardwerte der folgenden Variablen je nach Spielmodus gesetzt werden:

- learn with images:
local_two_player = false;
learn_with_images = true
rendering_enabled = true
trainings_mode_enabled = false
- learn with position:
local_two_player = false;
learn_with_images = false
rendering_enabled = false (optional, kann auf true gesetzt werden um das Spiel zu beobachten)
trainings_mode_enabled = false
- training:
trainings_mode_enabled = true
ansonsten wie die Einstellungen für **learn with images** oder **learn with position**

4.2 Battle-Pong Code und Algorithmen

4.2.1 Ball Physik

Für jeden Step wird der Vektor *move* berechnet.

Dafür wird der Bewegungsvektor *velocity* normalisiert und anschließend mit den Faktoren *speed* und *delta* multipliziert. Die Konstante *delta* steht für die Zeitangabe, wie lange der Step dauert und entspricht damit der im Settings Window(4.2) eingestellten Variable **Playtime/Step**(4.1.2).

Wann immer der Ball einen Spieler trifft und der aktuelle Faktor *speed* kleiner als **speed_max** ist wird *speed* um **speed_inkrement** erhöht. Dadurch wird der Ball bei jedem Treffer beschleunigt, bis die maximale Geschwindigkeit erreicht oder überschritten wurde. Ob die maximale Geschwindigkeit überschritten wird hängt davon ab, ob die Differenz zwischen **speed_min** und **speed_max** ein vielfaches von **speed_increment** ist.

Die Bewegung des Balls wird mit der Funktion **move_and_collide**[*move_and_collide()*] ausgeführt. Dieser Funktion wird der berechnete Vektor *move* übergeben. Damit die Kollisionen mit Nodes vom Typ `RigidBody2D`[`RigidBody2D`] erkannt werden, muss der zweite Übergabeparameter *infinite_inertia* auf *false* gesetzt werden.

Ist der Rückgabewert von **move_and_collide** nicht vom Typ *Nil* (analog zu *None* in Python), so ist es während der Bewegung zu einer Kollision gekommen.

Wurde ein Objekt vom Typ Wand oder dem Hindernis getroffen, so wird die Godot eigene **bounce()** Funktion [bounce Funktion] verwendet. Der Abprallvektor wird anhand des Normalenvektors des getroffenen Objekts bestimmt, sodass Eintrittswinkel und Austrittswinkel gleich sind.

Für den Abprallwinkel an einem Spieler wird der in Abschnitt 4.2.1.2 beschriebene Algorithmus verwendet.

4.2.1.1 Startwinkel

Damit sich der Ball zu Beginn des Spiels nicht jedes Mal in die selbe Richtung bewegt, wird diese mit Hilfe einer Zufallsfunktion bestimmt. Der x-Vektor, welcher angibt ob der Ball zu Spieler 1 oder Spieler 2 fliegt wird mit dem Wert 100 statisch angegeben. Dieser wird negiert, wenn die berechnete Zufallszahl (-1 bis 1) kleiner als 0 ist.

Für den y -Vektor wird zufällig ein Wert zwischen -200 und 200 gewählt.

Damit ist der Bewegungsvektor zum Rundenanfang bestimmt und kann einen Winkel zwischen $\pm 63,4^\circ$ von der horizontalen Mittellinie gesehen in Richtung einer der Spieler einnehmen. Die Grenzen in denen der Startvektor liegen kann sind in Abb. 4.8 eingezeichnet.

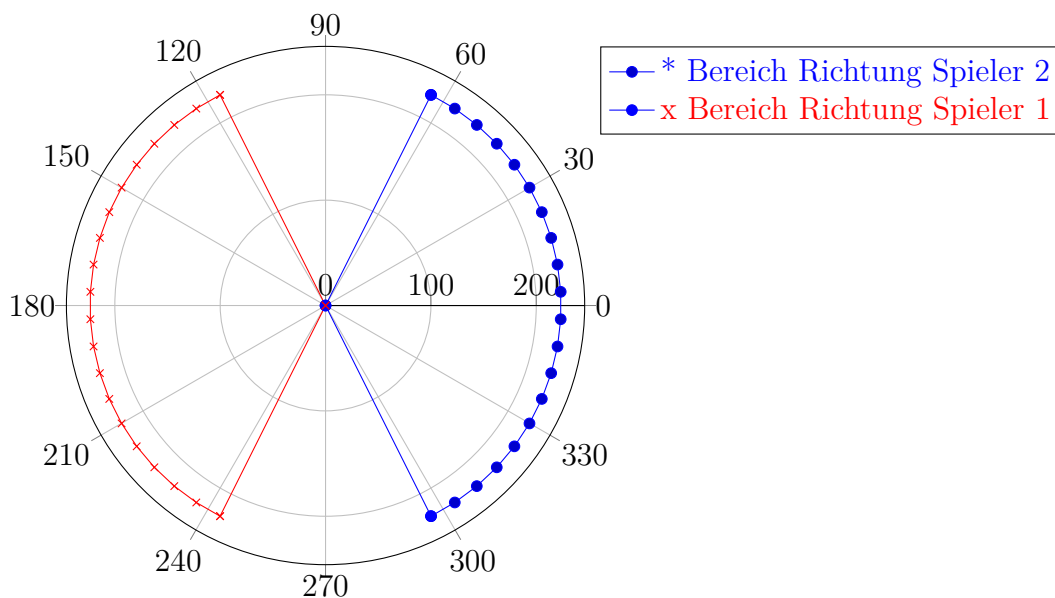


Abbildung 4.8: Ball Startwinkelbereich

4.2.1.2 Abprallwinkel an Spieler

Um interessantere Spielzüge zu ermöglichen wird der Abprallwinkel angepasst, je nachdem wo der Ball den Spieler getroffen hat.

Wir der Spieler exakt mittig getroffen, so entspricht der Eintrittswinkel dem Austrittswinkel.

Ist die absolute Entfernung des Aufprallpunktes (y-Position des Balls) zur vertikalen Mitte des Spielers größer 0 so wird der y-Anteil des Bewegungsvektors angepasst (Algorithmus: 1). Der berechnete Wert *bounce* wird zum y-Wert des Ball-Bewegungsvektors addiert. Dies

führt dazu, dass der Ball steiler oder flacher abgefälscht werden kann. Dadurch können schnelle Wechsel zwischen dem oberen und unteren Spielfeldrand herbeigeführt werden, was es erschwert den Ball abzuwehren.

Algorithmus 1 : Berechne y-Bewegungsvektor Anpassung

Data : $n \geq 0 \vee x \neq 0$
Result : $y = x^n$

```

1 bounceMax = 60
2 collision = yPlayer - yBall
3 halbePlayerHöhe = PlayerHöhe/2
4 halbeBallHöhe = BallHöhe/2
5 bounce = halbePlayerHöhe/bounceMax * |collision|
6 if collision > 0 then
7   | bounce = bounce * (-1)
8 else
9 end
```

4.2.2 Spieler Physik

Die Spieler können sich nur vertikal am jeweiligen Spielfeldrand bewegen. Dabei wird am Ende einer Bewegung überprüft, ob der Spieler außerhalb des erlaubten Bereichs ist und notfalls auf den nächstgelegenen Grenzwert zurückversetzt (obere oder untere Kante). Im Gegensatz zum Ball wird bei den Spielern keine Funktion zur Verschiebung verwendet, sondern die neue Position wird direkt gesetzt.

Je größer die Stepdauer gewählt wurde, desto weiter wird der Spieler in die entsprechende Richtung versetzt. Für feine Bewegungen ist demnach eine geringe Stepdauer notwendig.

Bei jedem Aufruf der Funktion *run(delta)* wird ein Bewegungsvektor *velocity* mit $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ erstellt. Je nach gedrückter Aktionstaste (siehe 4.1.4) wird der y-Anteil um 1 erhöht / verringert. Hat der Vektor eine Länge größer 0 wird er normalisiert und mit **speed** (siehe 4.1.2), sowie *delta* (Stepdauer) multipliziert.

Der so berechnete Bewegungsvektor wird zum Positionsvektor der Spieler addiert, womit die eigentlich Bewegung abgeschlossen ist.

Um sicherzustellen, dass sich Spieler nicht aus dem Spielfeld hinaus bewegen, wird die Funktion *clamp(value, min, max)* verwendet, welche überprüft ob ein Wert innerhalb der angegebenen Grenzen ist. Wird ein Grenzwert überschritten, so wird der entsprechende Grenzwert zurückgegeben, sonst der übergebene Wert.

4.2.3 Obstacle Physik

Bei jedem Step wird die *run()* Funktion des Obstacles aufgerufen. Dabei wird ein neuer Vektor *velocity* $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ erstellt. Der Wert 1 des y-Anteils sorgt dafür, dass sich das Objekt

nach unten bewegt. Dieser Vektor wird mit **speed** (siehe 4.1.2) und *delta* (Stepdauer) multipliziert. Der berechnete Bewegungsvektor wird dem aktuellen Positionsvektor aufaddiert. Ist die neue Position außerhalb des erlaubten Bereichs (läuft aus dem unteren Spielfeldrand heraus) wird die Position des Obstacles auf die Startposition am oberen Spielfeldrand gesetzt.

4.2.4 Entkopplung der Zeit

Standardmäßig werden Bewegungen über die **__process()** [`__process()`] oder auch **__physics_process()** [`__physics_process()`] Funktion verarbeitet. Die Funktion `__process()` wird bei jedem neuen Frame aufgerufen und dabei die Variable *delta* [Godot Delta] übergeben. Diese entspricht der Zeit in Sekunden seit dem letzten Aufruf der Funktion. Um eine framerate unabhängige Bewegungsgeschwindigkeit zu erreichen wird der Bewegungsvektor mit *delta* multipliziert. Da die Berechnungen der Physik meistens zeitkritisch sind wird die Funktion `__physics_process()` verwendet, denn diese wird konstant alle 1/60 Sekunden aufgerufen.

Um Spielzeit von der Realzeit zu entkoppeln wird die Funktion **run(delta)** verwendet, welche analog zur `__physics_process()` Funktion entwickelt wurde, jedoch manuell aufgerufen wird. Dadurch kann ein Step immer genau dann durchgeführt werden, sobald beide Agenten ihre nächste Aktion bekanntgegeben haben. Dies führt dazu, dass die tatsächlich verstrichene Zeit von den `run()` Funktionen (Berechnungsdauer) abhängt und nicht von der verstrichenen Zeit. Außerdem ist die im Spiel verstrichene Zeit konstant, was für die Arbeit mit Reinforcement Algorithmen notwendig ist.

4.2.5 Bewegungsreihenfolge

Da die Bewegungen manuell gestartet und nicht parallel durch die `__physics_process()` Funktion gesteuert werden, laufen diese sequentiell ab. Jedoch wird die Bewegungsreihenfolge bei einer hinreichend kleinen Stepdauer irrelevant, denn die Objekte werden nur um wenige Pixel verschoben.

Bevor die `run()` Funktionen ausgeführt werden, werden die Nodes aktiviert, was eine Bewegung erlaubt und nach abgeschlossener Bewegung werden diese wieder gestoppt. Dies ist für den Trainingsmodus notwendig, da die Spieler ansonsten über die Tastatureingabe einen größeren Bewegungszeitraum hätten.

Reihenfolge:

1. Spieler 1
2. Spieler 2
3. Ball
4. Hindernis (Obstacle), wenn aktiviert

4.2.6 Aufnahme und Übertragung der Screenshots

Bevor der Inhalt des Bildschirm aufgezeichnet werden kann, wird abgewartet bis das aktuelle Frame vollständig gezeichnet ist. Bei der Verwendung des Bildes ist darauf zu achten, dass das Bild auf dem Kopf steht und demnach vertikal gespiegelt werden muss. Nach der Spiegelung wird das Bild auf die, in den Einstellungen angegebene, Größe (siehe 4.1.2) reduziert. Bei der Reduzierung ist darauf zu achten, dass der richtige Interpolationsmodus ausgewählt ist. Standardmäßig ist die *nearest-neighbor interpolation* ausgewählt, wobei sich die Größe der Spieler und insbesondere des Balls von Bild zu Bild unterscheiden kann. Um dieses Problem zu verhindern wird die *bilinear interpolation* verwendet. Sobald das Bild die Zielgröße hat wird es je nach Einstellung in 8Bit Graustufen oder RGB mit 8Bit Farbtiefe konvertiert. Im Modus *L8* wird ein Pixel durch 1 Byte dargestellt und im Modus *RGB8* werden 3 Byte pro Pixel verwendet.

Für die Übertragung per Websocket muss das Bild in Base64 (siehe 3.6) konvertiert werden. Das konvertierte Bild aus Abb. 4.2 steht dann in der Variable `sg_saved_img_player_one` und ist bereit für die Einbindung in JSON.

Listing 4.2: Bildschirmaufnahme und Konvertierung

```
1 get_viewport().set_clear_mode(Viewport.CLEAR_MODE_ONLY_NEXT_FRAME)
2
3 # Wait until the frame has finished before getting the texture.
4 yield(VisualServer, "frame_post_draw")
5
6 var thumbnail = get_viewport().get_texture().get_data()
7 thumbnail.flip_y()
8 thumbnail.resize(width, height, Image.INTERPOLATE_BILINEAR)
9 if $"/root/GameSettings".image_rgb:
10     thumbnail.convert(Image.FORMAT_RGB8 ) # Farbe
11 else:
12     thumbnail.convert(Image.FORMAT_L8 ) # S/W
13
14 var img_array : PoolByteArray = thumbnail.get_data()
15 var base64_image = Marshalls.raw_to_base64(img_array)
```

4.2.7 "yield" - warum die Wartefunktionen notwendig sind

In Godot sind manche Funktionen doppelt. Die gedoppelten Funktionen werden durch ein *yielded* im Namen gekennzeichnet. Dieser Zusatz weist darauf hin, dass in diesen Funktionen *yield* verwendet wird. Bei *yield* handelt es sich um eine Wartefunktion. Das Programm wird an dieser Stelle pausiert bis das angegebene Ereignis eintritt. Die Wartefunktion ist notwendig um die Funktion anzuhalten, bis das letzte Frame fertig dargestellt wurde. Wird eine Funktion aufgerufen die mit Hilfe von *yield* auf ein Ereignis wartet, wird der aktuelle Status zurückgegeben und nicht der eigentliche Rückgabewert der Funktion. Daher muss

rückwirkend in allen Funktionen per `yield` auf den Funktionsabschluss gewartet werden, um auch tatsächlich den Rückgabewert zu erhalten.

Da diese Wartefunktionen die Programmablaufgeschwindigkeit merklich beeinflussen und es nur im Modus **learn with images** (siehe 4.1.1.2) notwendig ist auf die Bildschirmanzeige zu warten, wurden die Funktionen, die die Informationen für die Observation sammeln, gedoppelt. So werden nur dann `yield`-Funktionen verwendet, wenn Sie auch benötigt werden.

4.3 BattlePongAI

Die `BattlePongAI` enthält die Schnittstellenimplementierung und einen nicht lernenden Agenten, mit dem ein Spieler in Battle-Pong gesteuert werden kann. Das Projekt ist so aufgebaut, dass pro Spieler eine Python-Datei erstellt wird und jede einen Spieler steuert. Dies ist in sofern notwendig, da sich die Agenten in Godot mit jeweils einem eigenen Websocket Client anmelden. Dadurch bekommen sie individuelle Antworten. Dabei soll es für beide Spieler so aussehen als würden sie auf der gleichen Position spielen, was es ermöglicht die Agenten auszutauschen oder gegen sich selbst spielen zu lassen.

Damit die Nutzung möglichst einfach ist wurde die gesamte Kommunikation in der Klasse **Gym** gekapselt. Diese Klasse kümmert sich um den Auf- und Abbau der Kommunikation mit dem Websocket Server von Battle-Pong, sendet Actions und gibt die erhaltene Observation zurück. Dadurch muss innerhalb der Implementierung des Agenten nur die nächste Action ausgewählt und die anschließend erhaltene Observation ausgewertet werden.

4.3.1 API

Um die API zu initialisieren ist die folgenden Reihenfolge notwendig:

1. Erstellen einer Instanz von *Gym*
2. Mit `gym.connect_player()` die aktuelle Observation als Grundlage laden

4.3.1.1 Instanziierung von Gym

Initialisiert wird die API durch erstellen einer Instanz der Klasse **Gym**. Dabei wird die IP-Adresse des Computer übergeben auf dem Battle-Pong läuft, der verwendete Port und die Angabe ob Spieler 1 (**player_one**) oder Spieler 2 (**player_two**) gesteuert werden soll.

4.3.1.2 connect_player()

Mit der Funktion `connect_player()` meldet sich der Agent bei Battle-Pong an und erhält die initiale Observation. Mit dieser Observation kennt der Agent die aktuelle (Ausgangs-)Lage und kann seine erste Action planen.

4.3.1.3 reset()

Um das Spiel zu starten beziehungsweise nach einer beendeten Runde neu zu starten wird die *reset()* Funktion verwendet. Sie signalisiert Godot, dass eine neue Runde beginnen soll. Dies entspricht im Modus **Two Local Player** (siehe 4.1.1.1) dem Drücken der Leertaste. Wird die Funktion *reset()* aufgerufen führt das dazu, dass in Battle-Pong in der Datei *Main.gd* die Funktion **play()** aufgerufen wird.

Dabei wird die Variable *playing* auf *true* gesetzt, was dem Spiel signalisiert, dass ein Spiel läuft. Dementsprechend wird in der Observation die Variable *done* mit *false* zurückgegeben. Für den Fall, dass einer der Spieler die **Wins till end** (siehe 4.1.2) erreicht hat wird der Scorezähler zurückgesetzt und ein neues Spiel gestartet.

4.3.1.4 step(action)

Die Funktion *step(action)* nimmt einen Parameter vom Typ *ActionSpace* entgegen und schickt die Action an Battle-Pong. Damit in Battle-Pong die Action einem Spieler zugeordnet werden kann, wird der ausgewählte Spieler (Initialisierung Gym 4.3.1.1) als Präfix an die Action angehängt.

Ist in Godot der Step abgeschlossen und der Websocket Client hat die Observation empfangen, wird diese als JSON Objekt zurückgegeben.

ActionSpace Enum:

- nothing = 0
- up = 1
- down = 2
- training = 3

4.3.1.5 Aufbau Observation

Die empfangene Observation von Battle-Pong ist wie folgt aufgebaut:

- observation:
Information über Position der Spieler, sowie Position und Bewegungsvektor des Balls
Wenn der Trainingsmodus Aktiv ist steht in *TrainingAction* die auf der Tastatur gedrückte Taste.
- reward:
Erzielt ein Spieler einen Punkt steht der Reward für den Spieler auf +1 und bei dem anderen Spieler auf -1
Ansonsten ist reward immer 0

- **done:**
Gibt an ob das Spiel noch läuft oder ob die *reset()* Funktion (siehe 4.3.1.3 aufgerufen werden muss
- **info:**
In *Score* stehen die gesammelten Punkte
- **screenshot:**
Enthält den, nach Abschluss des Steps, aufgezeichneten Screenshot
 - **image:**
In Base64 kodiertes Bytearray - Das eigentliche Bild
 - **size:**
Byteanzahl des Bildes - zur Überprüfung
 - **format:**
Das Format des Bildes
0 = L8 (8 Bit Graustufen)
4 = RGB8 (Farbe in 8 Bit Farbtiefe)

Listing 4.3: Empfange Observation in JSON

```

1 {
2   'observation': {
3     'PlayerOne': {
4       'X': 52,
5       'Y': 300,
6       'TrainingAction': ''
7     },
8     'PlayerTwo': {
9       'X': 972,
10      'Y': 300,
11      'TrainingAction': ''
12    },
13    'ball': {
14      'velocity': {
15        'X': -139.04834,
16        'Y': 265.829956
17      },
18      'position': {
19        'X': 512,
20        'Y': 300
21      }
22    }
  }

```

```

23     },
24     'reward': {
25         'PlayerOne': 0,
26         'PlayerTwo': 0
27     },
28     'done': True,
29     'info': {
30         'PlayerOneScore': 0,
31         'PlayerTwoScore': 0,
32         'screenshot': {
33             'image': 'Hier stehen 6000 Byte in Base64',
34             'size': 6000,
35             'width': 100,
36             'height': 60,
37             'format': 0
38         }
39     }
40 }

```

4.3.2 Beispiel Agent RoboCat

Der Agent RoboCat ist ein simpler nicht lernender Agent, welcher versucht die Höhe (y-Achse) des Balls zu halten.

Trotz Seines schlichten Aufbaus stellt er für lernende Agenten eine große Herausforderung dar. Solange der vom RoboCat gesteuerte Spieler in der Lage ist dem Ball zu folgen ist dieser nicht zu besiegen. Dies bedeutet, dass das Spiel entweder so lange laufen muss bis die Geschwindigkeit des Balls die des Spielers überschreitet oder der Ball steil genug gespielt wird.

Durch den schnellen Wechsel zwischen dem oberen und unteren Spielfeldrand ist der vom Agenten gesteuerte Spieler bereits früher nicht mehr in der Lage dem Ball zu folgen. Die effektive Geschwindigkeit des Balls verändert sich bei einem schnellen Wechsel zwar nicht, allerdings macht es einen Unterschied, ob sich der Ball direkt auf den Spieler zu bewegt oder sich nahezu vertikal bei gleicher Geschwindigkeit bewegt.

4.4 Kommunikation

Die Kommunikation ist über Websockets (siehe 3.4) realisiert worden, da diese die Vorteile einer TCP Übertragung besitzen und dennoch schnell sind. Ein Beispiel hierfür wäre die Überprüfung, ob die Übertragung vollständig erhalten wurde. Zudem sind sowohl in Go-dot, als auch in Python, fertige Socketimplementierungen vorhanden. Dadurch konnte auf eine getestete und stabile Lösung aufgebaut und Entwicklungszeit verkürzt werden. Zudem musste keine Zeit für die Fehlersuche bei der Datenübertragung aufgewendet werden.

Um beide Agenten separat ansprechen zu können, schickt der Client/Agent die Bezeichnung des Spielers mit. Dadurch kann bei Ankunft der Daten die ClientID einem Spieler zugeordnet und jedem Spieler eine spezifische Observation geschickt werden.

Beim Training der lernenden Algorithmen kam die Fragestellung nach einem Seitenwechsel der Agenten auf und dafür musste die Observation für beide Spieler so aussehen, als wären Sie auf der gleichen Seite.

Durch das direkt ansprechen der Spieler konnte diese Anforderung umgesetzt werden.

4.4.1 BattlePongAI \rightarrow Battle-Pong

Die gesendeten Pakete vom Client an den Server enthalten immer zwei Angaben:

- Spielerbezeichnung
 - *player_one*
 - *player_two*
- Action
 - Bsp.: *player_one_up*

Mit der Spielerbezeichnung kann die temporäre Websocket Client ID dem Spieler / dessen Agenten zugeordnet werden.

Die Action wird in Godot hinterlegt und wenn für beide Spieler der nächste Step angegeben ist wird dieser ausgeführt.

4.4.2 BattlePongAI \leftarrow Battle-Pong

In Battle-Pong wird die zusammengesetzte Observation mit Hilfe der *JSON.print()* Funktion serialisiert und anschließend an die angemeldeten Clients geschickt. Da die Clients bei jeder Anmeldung eine neue ID haben wird auf die gespeicherte Client ID der Spieler zugegriffen. Dadurch kann den Spielern eine individuelle Antwort gesendet werden.

4.5 Installationsanleitung

4.5.1 Godot

1. Repository als zip Downloaden [Gitlab Repository]
 - Master Branch für den aktuellsten Stand
 - Release, für stabile und getestete Version
2. Archiv entpacken und in das gewünschte Projektverzeichnis kopieren
3. Download der aktuellen Godot Engine (Windows - Standard version, 64-Bit)[Godot Download]

4. Entpacken des zip-Archivs
5. Ausführen der Anwendung
6. Im Projekt Manager auf **Import** klicken
 - **Project Path:** => *Entpacktes battle-pong Projektverzeichnis/Battle Pong/project.godot* auswählen
7. **Import & Edit** klicken

4.5.2 Python

1. Ordner BattlePongAI in beliebiger IDE öffnen
2. Importierte Pakete installieren

4.5.3 Test

1. Battle-Pong in Godot starten
2. Haken bei **Enable rendering** setzen
3. **2 RE with Position** klicken
4. BattlePongAI main.py starten
5. BattlePongAI main_player_two.py starten
 - => Das Spiel wird gestartet und die beiden RoboCats (Default-Agents in Python) spielen gegeneinander.
 - Dadurch, dass das Rendern eingeschaltet wurde, kann das Spiel beobachtet werden.

5 Ergebnisse

5.1 Godot

Prinzipiell ist der Einstieg in Godot sehr einfach, da die Dokumentation sehr ausführlich und leicht zu verstehen ist. Zudem gibt es online zahlreiche YouTube Videos, Foren und Tutorials, die eine schnelle Problemlösung möglich machen. Die große Nähe der Programmiersprache GDScript zu Python erleichtert den Einstieg ebenfalls.

Dennoch erfordert das Programmieren eines Spiels ein Umdenken, da viele Aktionen parallel ablaufen. Eine weitere Voraussetzung ist der sichere Umgang mit Vektorrechnung.

Insgesamt ist Godot für Projekte dieser Art von Spielen zu empfehlen und für Einsteiger geeignet.

5.2 Python

Die Implementierung eines Agenten in Python war simpel und erfordert wenig Code. Die, auf dem Gym toolkit basierende, API ist benutzerfreundlich, da es kaum Einarbeitungszeit bedarf. Vorteilhaft ist auch die Kommunikation per Websocket. Die Kommunikation per Websocket ist vorteilhaft, da Agent und Spiel nicht zwangsläufig auf dem gleichen Computer laufen müssen. Sie können über das Netzwerk verteilt sein.

Dies macht das

6 Fazit (und Ausblick)

Während der Dauer des Projekts hat sich die Verwendung der Websockets als Übertragungsmedium als gute Wahl herausgestellt, da die Übertragungszeit, im Verhältnis zur Rechenzeit der Agenten, zu vernachlässigen ist.

Überdies ist die Implementierung der Agenten dank der Websockets nicht an Python gebunden, sondern es kann eine beliebige Programmiersprache gewählt werden, solange sie die Kommunikation per Websockets unterstützt. Das Lernen über das Internet ebenfalls möglich allerdings muss die Übertragungszeit beachtet werden, da die für das Lernen notwendige Zeit zu groß werden kann.

Es hat sich gezeigt, dass die Aufnahme des Screenshots bei einfachen Oberflächen ineffizient ist und eine Generierung aus Positionsdaten sinnvoller wäre. Eine Schwachstelle ist derzeit die Geschwindigkeit, wenn mit Bildern gelernt werden soll, denn das Rendern eines Frames ist sehr zeitaufwändig. Auch ist die Hardwareabhängigkeit nicht zu unterschätzen, da diese die Ausführungsdauer eines Steps stark beeinflusst. Je höher die FPS (Frames per Second) Anzahl, desto geringer ist die Berechnungsdauer des Steps. Die Abhängigkeit kann minimiert werden, wenn das Bild nicht aufgenommen, sondern aus den Positionsangaben erzeugt wird. So könnte man auch das Problem der Seitenabhängigkeit der Agenten eliminieren.

Da die Spieler unterschiedliche Farben haben ist ein Seitenwechsel derzeit ohne Bildbearbeitung oder Anpassung der Farben im Code des Spiels nicht möglich.

Ein richtiger Headless Modus wäre bei generierten Bildern ebenfalls denkbar, da für das Spiel dann kein GUI (Graphical User Interface) mehr benötigt wird.

Ein neuer Modus, bei dem sammelbare Items die Spielergröße oder die Anzahl der Bälle verändern wäre ebenfalls denkbar. Vor diesem Hintergrund ist das Projekt erfolgreich, hat jedoch noch viel Potential und wird auch in Zukunft weiterentwickelt.

Über GitLab [Gitlab Repository] können Fehler, Verbesserungsvorschläge oder Featureanfragen gemeldet werden, welche dann dem Projekt implementiert werden.

Beim Schreiben dieser Hausarbeit sind sowohl in Battle-Pong, als auch in BattlePongAI Stellen aufgefallen, die verbessert oder vereinfacht werden können.

Abbildungsverzeichnis

3.1	Atari Pong	5
4.1	Battle-Pong Hauptmenü	11
4.2	Battle-Pong Game Settings	13
4.3	Godot - Menüzeile - Project	16
4.4	Godot - Project Settings - Input Map	17
4.5	Godot - Settings - Edit Input Map	17
4.6	Godot - Menüzeile - Project	18
4.7	Godot - Project Settings - General - Run	18
4.8	Ball Startwinkelbereich	20

Tabellenverzeichnis

3.1 The Base64 Alphabet 9

Liste der Algorithmen

1	Berechne y-Bewegungsvektor Anpassung	21
---	--	----

Listings

4.1	Generierte Settingsdatei: data.json	15
4.2	Bildschirmaufnahme und Konvertierung	23
4.3	Empfange Observation in JSON	26

Literaturverzeichnis

- [Godot] "Godot Engine: Startseite <https://godotengine.org/>; abgerufen am 9. April 2021."
- [Godot Download] "Godot Engine: Downloadseite <https://godotengine.org/download>; abgerufen am 9. April 2021."
- [Your first Game] "Godot: Das erste eigene Spiel https://docs.godotengine.org/en/3.2/getting_started/step_by_step/your_first_game.html; abgerufen am 9. April 2021."
- [Step-By-Step Anleitung] "Godot: Schrittweise Einführung in die Spiele Engine https://docs.godotengine.org/en/3.2/getting_started/step_by_step/index.html; abgerufen am 9. April 2021."
- [Base64 Encoding] "IETF.org - RFC 2045: Base64 Content-Transfer-Encoding <https://tools.ietf.org/html/rfc2045.html#section-6.8>; abgerufen am 7. April 2021."
- [Godot Delta] "KidsCanCode - Understanding 'delta' https://kidscancode.org/godot_recipes/basics/understanding_delta/; abgerufen am 7. April 2021."
- [_process()] "Godot Documentation - void _process (float delta) virtual https://docs.godotengine.org/en/3.2/classes/class_node.html#class-node-method-process; abgerufen am 7. April 2021."
- [_physics_process()] "Godot Documentation - void _physics_process (float delta) virtual https://docs.godotengine.org/en/3.2/classes/class_node.html#class-node-method-physics-process; abgerufen am 7. April 2021."
- [Interview N. Bushnell] "Heise - Interview mit Nolan Bushnell, Erfinder von Pong und Atari-Gründer von 10.11.1998 <https://www.heise.de/tp/features/Interview-mit-Nolan-Bushnell-Erfinder-von-Pong-und-Atari-Gruener-10111998.html>; abgerufen am 7. April 2021."
- [Formate] "Godot: Image Formate https://docs.godotengine.org/en/3.2/classes/class_image.html#enum-image-format; abgerufen am 9. April 2021."

- [move_and_collide()] "Godot: move_and_collide() Funktions Dokumentation https://docs.godotengine.org/en/3.2/tutorials/physics/using_kinematic_body_2d.html#move-and-collide; abgerufen am 9. April 2021."
- [RigidBody2D] "Godot: RigidBody2D Dokumentation https://docs.godotengine.org/en/stable/classes/class_rigidbody2d.html#class-rigidbody2d; abgerufen am 9. April 2021."
- [bounce Funktion] "Godot: bounce() Funktion Dokumentation https://docs.godotengine.org/en/stable/classes/class_vector2.html#class-vector2-method-bounce; abgerufen am 9. April 2021."
- [Gitlab Repository] "Battle-Pong GitLab Repository <https://gitlab.cvh-server.de/faust/battle-pong>; abgerufen am 9. April 2021."
- [Godot Programmiersprachen] "Godot: Programmiersprachen <https://docs.godotengine.org/en/3.2/tutorials/scripting/index.html#programming-languages>; abgerufen am 9. April 2021."