Computerlinguistik-Projekt: Simple audio recognition of speech commands for moonlanding game

Dennis Binz, Nathalie Elsässer, Julia Karst,
Sarah Ondraszek, Till Preidt

Computerlinguistik – Projektseminar

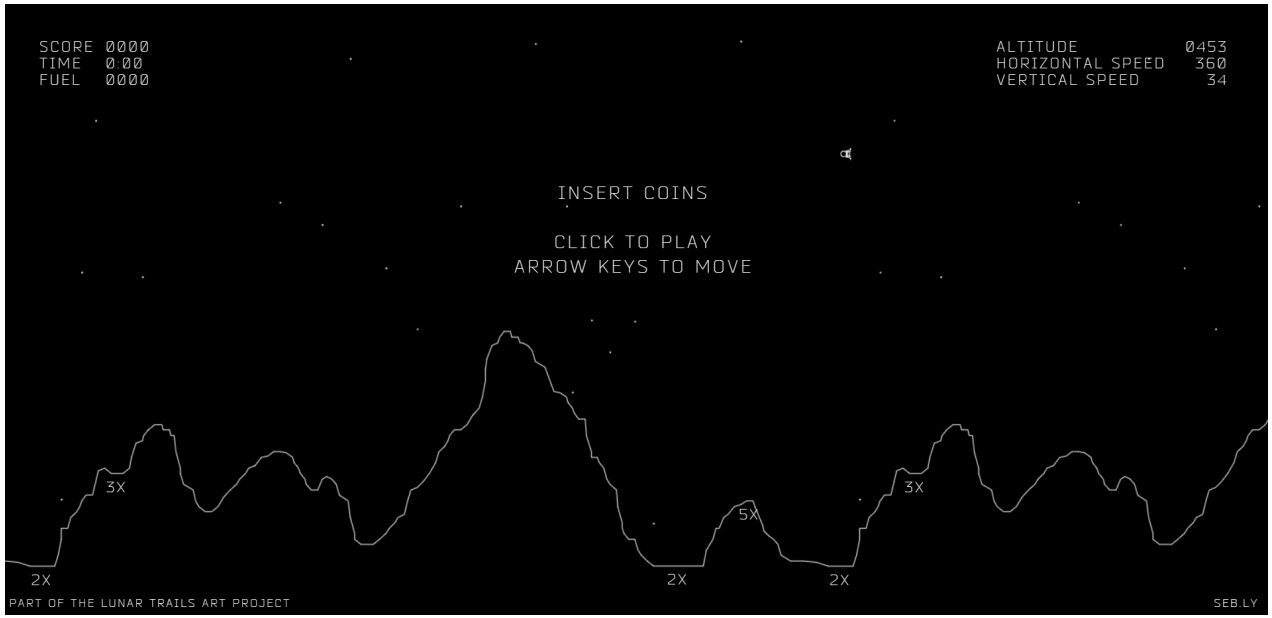
Sommersemester 2021
13.07.2021

Gliederung

- Idee und Ziele
- Anpassung und Training des Modells
- Test-Input und Test des Modells
- Programmierung des Spiels
- Mikrofon-Input
- Zusammenführung
- Fazit

Idee

- per Sprachbefehle steuerbare Version des Spiels „Moonlander“
  $\rightarrow$ Rakete sicher auf Mond landen
- Programmierung in Python 3.8
- Training eines Sprachmodells zur Spracherkennung



`http://moonlander.seb.ly/`

gesetzte Ziele

- lauffähiges Spiel
- Sprachbefehle zur Steuerung des Spiels
- Sprachbefehle während des Spiels einsprechen
- automatische Erkennung von Sprachbefehlen
- Analyse der Sprachbefehle durch neuronales Netz
- Sprachbefehle sollen richtig erkannt werden
- Spiel soll zum gesprochenen Befehl zugehörige Aktion ausführen
- Spiel soll ohne große Verzögerungen spielbar sein

Wahl des Modells

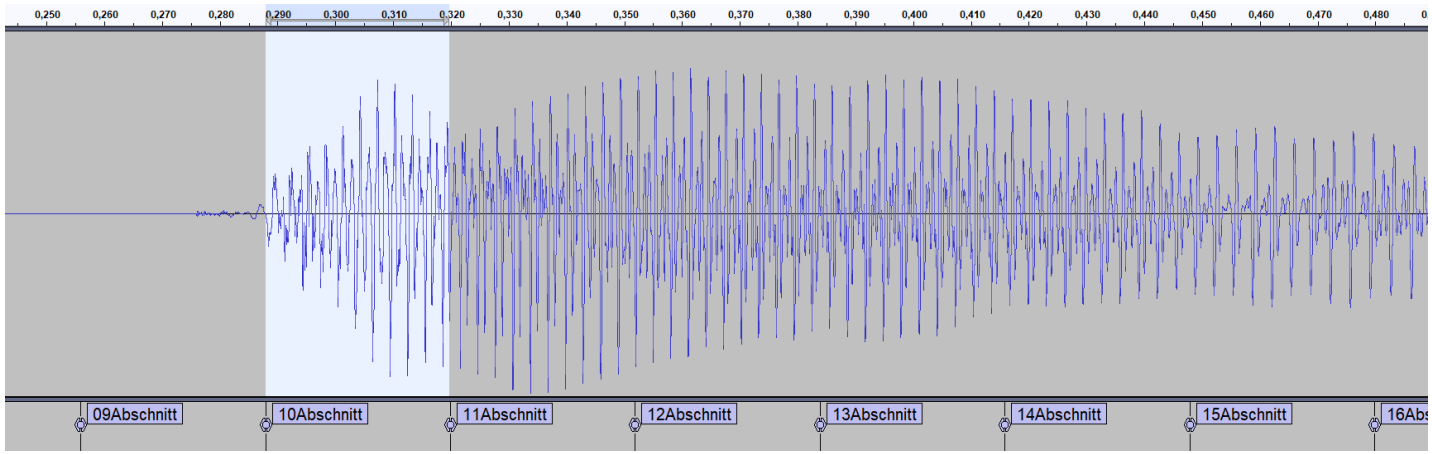
- Vergleich verschiedener verfügbarer Skripts, TensorFlow und MFCCs
- Wahl nach ökonomischem Prinzip und Nutzbarkeit
- Wahl des Datensatzes: Zunächst kleiner DS, dann großer DS
- Anpassung und Umformatierung des MFCC-Skripts in autarke Komponenten

Formatierung der Daten

1. rohe WAVE-Daten in Abschnitte unterteilen
2. Diskrete Fourier-Transformation
3. MFCCs als Arrays speichern, ggf. Padding
   $\rightarrow$ librosa

Formatierung der Daten

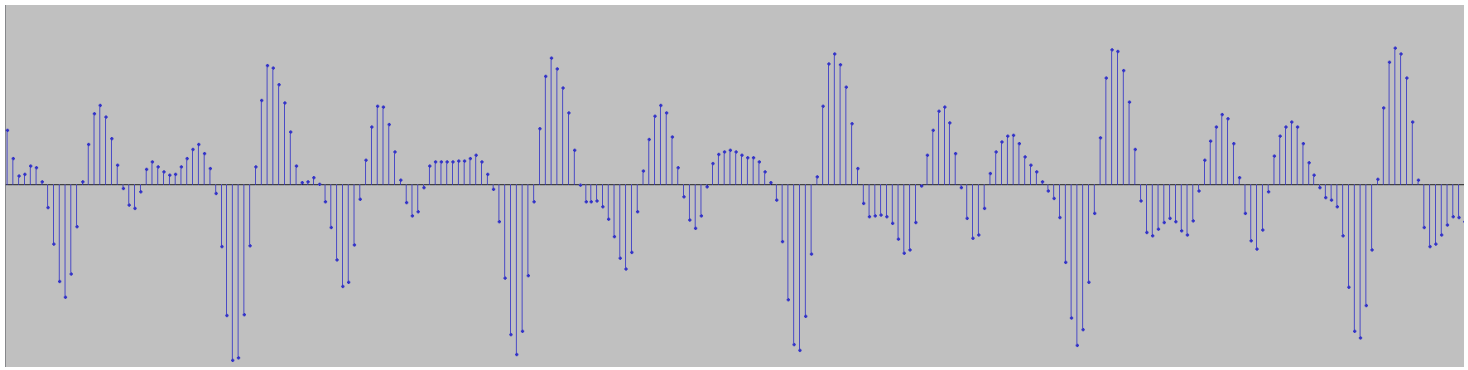
1. rohe WAVE-Daten in Abschnitte unterteilen
2. Diskrete Fourier-Transformation
3. MFCCs als Arrays speichern, ggf. Padding
   $\rightarrow$ librosa



32-millisekündige Abschnitte der WAVE-Datei

Formatierung der Daten

1. rohe WAVE-Daten in Abschnitte unterteilen
2. Diskrete Fourier-Transformation
3. MFCCs als Arrays speichern, ggf. Padding
   $\rightarrow$ librosa



zeitdiskretes Signal

Vorprozessierung/Training/Feintuning

- Vorprozessierung mittels Umwandlung der WAVE-Dateien in MFCCs
- MFCCs werden für Commands als Numpy-Arrays gespeichert
- Numpy-Arrays wiederum werden aufgesplittet (für Trainingsinput) und *randomisiert*
- Training mit TensorFlow/Keras, Modell wird zwischengespeichert
- Parameteranpassung - Feintuning
- Methoden für Speech Command Predictions
- Test via Pipeline auf gewählten Daten, siehe nächste Folien

Test-Daten generieren

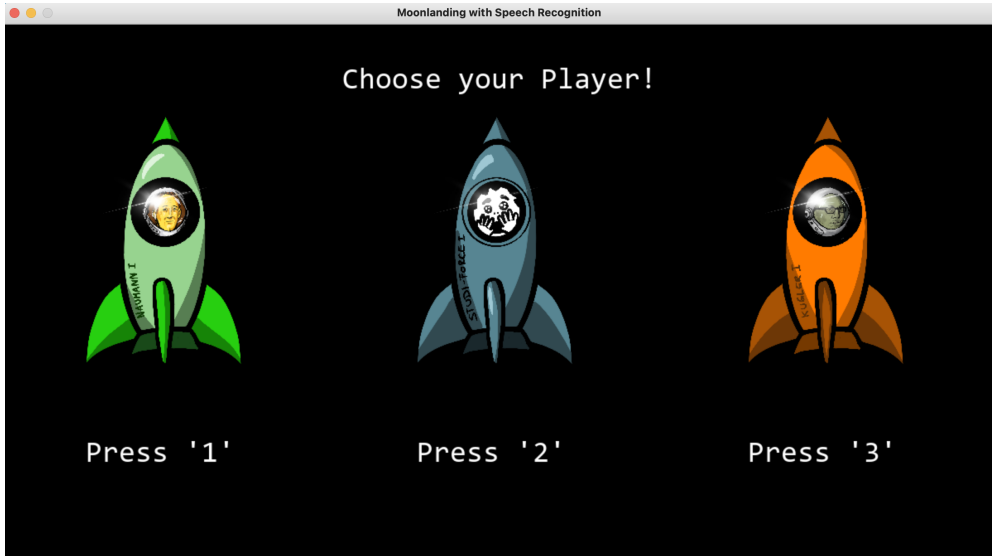
- Erstellung von Testbefehlen mit Audacity
- Format:
  - 16 kHz
  - mono
  - Samplingtiefe: 16-bit
    Bitrate: 256 kBit/s
- 80 Dateien, um Audioformat zu testen
- 168 Dateien, um Länge zu testen
- endgültiges Testset: 247 1-sekündige Dateien
- 162 Dateien „left“ von verschiedenen Freiwilligen

Test des Modells

- Modell aus Datei laden
- Testen mit der Pipeline auf Menge von Daten möglich

Spiel-Programmierung

- mithilfe von Pygame
- Spielerauswahl, Startbildschirm, Spiel und Endbildschirm
- gewählte Rakete beeinflusst Schwierigkeitsgrad

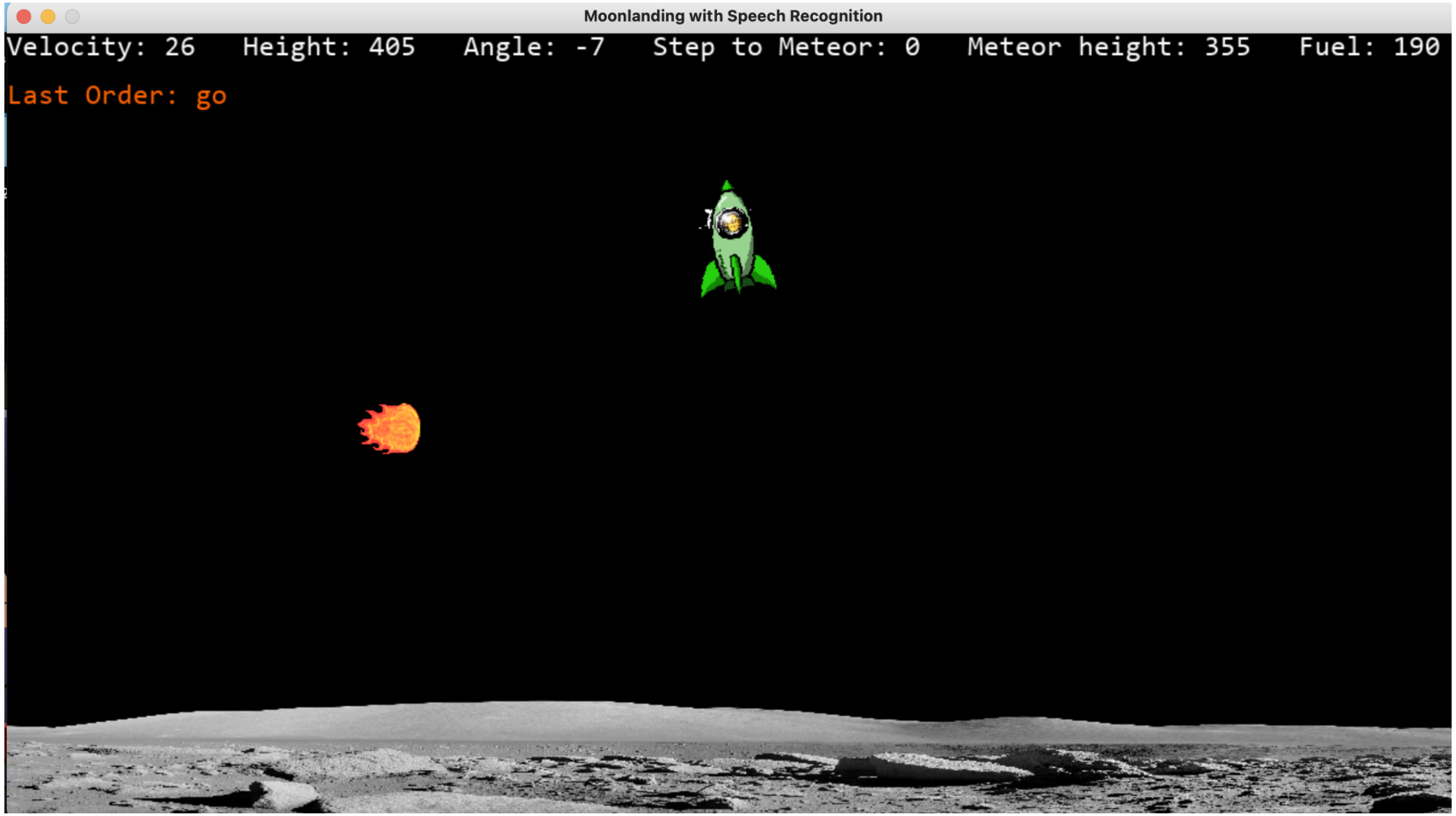


Spielerauswahl

Spiel-Programmierung

- Sprachsteuerung bis auf die Spielerauswahl überall nutzbar
- Angaben zu Rakete und Meteor
- Flaggensimulation und bei Absturz Explosion
- eigene Grafiken (Raketen, Flaggen, Explosion) von S. O.

Spiel-Programmierung



Moonlanding-Spiel

Spiel-Programmierung

Klasse „Player“ für die Rakete:

- Tastendruck wird in update-Funktion verarbeitet
  → Schub abhängig von Schwierigkeitsgrad
- Winkel wird in angle_change-Funktion zufällig verändert
  → abhängig von Schwierigkeitsgrad

Klasse „Meteor“:

- zufällige Schritte bis zur Kollision (5 - 15)
- zufällige Kollisionshöhe (100 - 400)

Weitere Klassen für die Grafiken: Moon, Microphone, Explosion, Flag

Spiel-Programmierung

- Wahl zwischen Pfeiltasten und Sprachsteuerung
- Sprachsteuerung durch Drücken der Leertaste aktiviert: Befehl wird drei Sekunden lang aufgenommen
- Simulation des Tastendrucks
- erfolgreiche Landung:
  - Meteor ausweichen
  - auf Treibstoff achten
  - Geschwindigkeit < 10
  - Neigungswinkel in [-6; 6]

Spiel-Programmierung

Der letzte Befehl wird auf dem Bildschirm angezeigt.
Mögliche Befehle:

- „go“ oder 'Enter': Spiel starten (Startbildschirm)
- „up“ oder '↑': Schub geben
- „down“/„go“ oder '↓' : fallen lassen
- „left“ oder '←': nach links neigen
- „right“ oder '→': nach rechts neigen
- „stop“: Spiel beenden
- „yes“ oder 'R': zurück zur Spielerauswahl (Endbildschirm)
- „no“ oder 'Esc': Spiel beenden (Endbildschirm)

Audio-Recording

Durch Drücken der Leertaste wird das Programmgeschehen in das Skript „input_key_ctrl_micr.py“/„Input-Skript“ verlagert.
Ablauf:

- Aufruf der Funktion „record_order“
- Importierung der Packages „sounddevice“ für Zugriff auf Mikrofon und „scipy.io.wavfile“ zur Erzeugung einer Wave-Datei
- Zugriff erfolgt auf Standard-Mikrofon (im Programm nicht änderbar)

Audio-Recording

Nutzung der Funktion „rec“ aus sounddevice zur Aufnahme.
Parameter:

- Abtastrate: 16 kHz
- Dauer: (noch) drei Sekunden
- eine Tonspur / mono

- Die Parameter müssen mit den Eigenschaften der Trainingsdaten für das NN übereinstimmen.
- Funktion „wait“ aus sounddevice verhindert Threading, hält Programm an, bis Aufnahme beendet.

Audio-Recording

Nutzung der Funktion „write“ aus scipy.io.wavfile zum Schreiben der Aufnahme in eine Wave-Datei.

- Speicherort: Projektordner „audio_recognition_moonlanding“
- Speicherpfad konstant, jede neue Aufnahme überschreibt die ursprüngliche Wave-Datei

Abschließende Schritte:

- Übergabe an Skript „cut_1sec.py“ zum Herausfiltern von Intensitätsmaxima und Kürzen
- Übergabe an Funktion „make_single_prediction“ aus Skript „predict.py“ zur Vorhersage des gesprochenen Befehls

Trimmen des Audioinputs

- 3-sekündige Audiodatei auf 1 s trimmen
- Verwendung von `pydub`
- Messen der Lautstärke in *dBFS*
  $\rightarrow$ lauteste Stelle einer Datei soll herausgeschnitten werden
- Test von verschiedenen Ansätzen
  $\rightarrow$ 960 Testdateien, Auswertung nach jedem Ansatz
  $\rightarrow$ festhalten der Ergebnisse in Tabelle

Trimmen des Audioinputs

Endgültiger Ansatz:

1. WAVE-Datei wird normalisiert
2. nicht-stille Audioparts werden identifiziert
3. lautester nicht-stiller Part wird identifiziert
   a. ist der Part >1 s:
      kürzen
   b. ist der Part <1 s:
      mehr Audiomaterial dazunehmen
4. alte Datei überschreiben

Trimmen des Audioinputs

Endgültiger Ansatz:

gesagt\predicted	left	right	up	down	yes	no	go	stop	∑
left	60,00%	0,00%	16,67%	0,00%	6,67%	10,00%	6,67%	0,00%	100,00%
right	20,00%	63,33%	10,00%	0,00%	3,33%	0,00%	3,33%	0,00%	100,00%
up	6,67%	3,33%	80,00%	0,00%	3,33%	3,33%	0,00%	3,33%	100,00%
down	6,67%	6,67%	10,00%	73,33%	0,00%	0,00%	0,00%	3,33%	100,00%
yes	6,67%	3,33%	0,00%	6,67%	76,67%	0,00%	3,33%	3,33%	100,00%
no	6,67%	0,00%	13,33%	0,00%	0,00%	43,33%	36,67%	0,00%	100,00%
go	3,33%	0,00%	13,33%	0,00%	0,00%	0,00%	83,33%	0,00%	100,00%
stop	6,67%	0,00%	23,33%	0,00%	0,00%	0,00%	3,33%	66,67%	100,00%

Erreichte Akuratesse: 68,33%

Audio-Recording und Prediction

- Audio aufnehmen
- Skript der Audioaufnahme ruft Trimmen des Audioinputs auf
- Audiodatei wird überschrieben
- Audiodatei wird an Modell übergeben und Befehl wird predictet
- Rückgabe des erkannten Befehls als String

Prediction und Spiel

Hauptteil und Menüs des Spiels laufen in while-Schleifen, warten auf Benutzereingabe:

- Erweiterung der vordefinierten Key_Events um K_SPACE
- Sprachsteuerung an dieses Event (also die Leertaste) koppeln
- Tastaturereignisse werden separat behandelt
- Ermöglicht Tastensteuerung, Sprachsteuerung, gemischte Steuerung (Spieler entscheidet mit jedem Zug)

Prediction und Spiel

Hauptteil und Menüs des Spiels laufen in while-Schleifen, warten auf Benutzereingabe:

- Erweiterung der vordefinierten Key_Events um K_SPACE
- Sprachsteuerung an dieses Event (also die Leertaste) koppeln
- Tastaturereignisse werden separat behandelt
- Ermöglicht Tastensteuerung, Sprachsteuerung, gemischte Steuerung (Spieler entscheidet mit jedem Zug)

$\rightarrow$ NN wird nur im Rahmen der Sprachsteuerung ins Spiel eingebunden
$\rightarrow$ Nützlich, falls Hardware-Probleme bei der Sprachsteuerung auftreten

Prediction und Spiel

Sprachsteuerung der Rakete im Spiel und der Menüs (Start- und End-Bildschirm) wurden unterschiedlich umgesetzt.
→ Zunächst Beschreibung der verbalen Raketensteuerung
→ Danach Sprachsteuerung in Menüs

Prediction und Spiel

Verbale Raketensteuerung:

- Nutzer drückt Leertaste
- Zunächst noch keine Aktion in Bezug auf die Spielobjekte ausgelöst
- boolsche Steuervariablen werden aktualisiert
  $\rightarrow$ Pausiert alle Spielelemente, die nicht von der Art der Benutzereingabe abhängig sind (z.B. Schrittzähler des Meteors)

Prediction und Spiel

Verbale Raketensteuerung:

- ...
- while-Schleife läuft zu Ende und beginnt von vorne (Spiel kann an dieser Stelle nicht terminieren)
- vor Tastenabfrage erfolgt Mikrofonabfrage
- Aufruf von record_order im Input-Skript, Audio-Input-Generierung

Prediction und Spiel

Verbale Raketensteuerung:

- ...
- record_order liefert den vom NN erkannten Befehl als String zurück
- Analyse des Strings via if-else
- In diesem Kontext relevante Keywords:
  „right“, „left“, „up“, „down“, „go“, „stop“
- wurde ein Keyword erkannt, Weitergabe des Strings an simulate_key_press im Input-Skript

Prediction und Spiel

Verbale Raketensteuerung:

- ...
- simulate_key_press nutzt String selbst als Parameter, simuliert entsprechenden Tastendruck
  $\rightarrow$ nutzt Funktion „keyDown“ aus Package pyautogui
- Rückkehr in while-Schleife des Spiel-Skripts
- Tastendruck-Simulation löst Key_Event in Schleife aus
- Spiel ruft entsprechende Funktionen auf

Prediction und Spiel

Verbale Raketensteuerung:

- ...
- in update-Funktion des Raketen-Objektes wird auf Simulation eines Tastendrucks geprüft
  → Simulierte Taste wird mittels Aufruf von release_key im Input-Skript wieder gelöst
  → release_key nutzt Funktion „keyUp“ aus Package pyautogui
- Kreislauf beginnt von vorne (sofern Spiel nicht terminiert)

Prediction und Spiel

Verbale Raketensteuerung: Warum so kompliziert?

- Die update-Funktion der Rakete erwartet als Parameter ein Key_Event $\rightarrow$ das Event, welches bei einem Tastendruck ausgelöst wird, muss direkt übergeben werden
- Deshalb werden Tastaturereignisse bei der Sprachsteuerung simuliert, um diese Events bei einem zweiten Aufruf der while-Schleife automatisch auszulösen
- Sprachsteuerung in Menüs konnte deutlich einfacher gehalten werden

Prediction und Spiel

Verbale Menü-Steuerung:

- Betrifft nur den Start- und End-Bildschirm
- Nutzer drückt Leertaste, Aufruf von record_order
- zurückgegebener Befehl wird direkt an Menüsteuerung gekoppelt
  $\rightarrow$ genau wie die bei Tastendrücken ausgelösten Key_Events
- relevante Keywords im Start-Bildschirm: „go“, „stop“
- relevante Keywords im End-Bildschirm: „yes“, „no“, „stop“

Modulverwaltung

- Umwandlung der einzelnen Skript-Ordner in Packages/Module
- Einfügen einer Variable is_game für Modulverwaltung
- Festlegen von Systemvariablen

Problematiken

- Probleme je nach Nutzer unterschiedlich bzw. nicht vorhanden:
  - Steckenbleiben auf dem Startbildschirm
  - keine Aktualisierung der Bilder
  - Starten vom Terminal aus nicht möglich
- **clock.tick()**: zur Aktualisierung der Bilder verwendet
- **pynput.keyboard**: für die Simulation der Tastendrücke verwendet
- Lösung: Ersetzung von *pynput.keyboard* durch *pyautogui*

Erreichte Ziele

- lauffähiges Spiel ✓
- Sprachbefehle zur Steuerung des Spiels ✓
- Sprachbefehle während des Spiels einsprechen ✓
- automatische Erkennung von Sprachbefehlen ✖
- Analyse der Sprachbefehle durch neuronales Netz ✓
- Sprachbefehle sollen richtig erkannt werden ✿
- Spiel soll zum gesprochenen Befehl zugehörige Aktion ausführen ✓
- Spiel soll ohne große Verzögerungen spielbar sein ✓

Zusammenfassung

- einige Problemlösungen benötigten mehr Zeit als vorher absehbar
- continuous speech auf Grund von Zeitmangel nicht mehr umsetzbar
- höhere, mikrofonunabhängige Erkennungsrate wäre wünschenswert gewesen
- die meisten Ziele konnten umgesetzt werden