

Pacman AI Dokumentation

Hannes Höttinger ¹

¹*FH Technikum Wien, Game Engineering und Simulation, Wien, AUT*

Abstract — Dieses Dokument dient als Dokumentation für die implementierte Pacman AI und als eine kurze Beschreibung der Programmausführung.

1 Aufgabenstellung

Zu implementieren ist die KI des Spiels Pacman. Die logischen Züge der Geister sind gegeben. Folgende Funktionalitäten müssen implementiert werden:

Input

1. GameMap einlesen, mit dem aktuellen GameState (pursuit/patrol/escape)
2. Jeder Ghost entspricht einem gewissen Char-Array → ein Buchstabe pro Richtung
3. GameMap in geeigneter Form speichern

Verarbeitung

1. Sensorik liest Informationen aus Speicher
2. Berechnet die aktuelle Bewegungs- und Blickrichtung
3. Alte Position wird mit Leerzeichen ersetzt

Output

1. Zeichne neue GameMap mithilfe der zur Verfügung gestellten Tile-Sets
2. Ersetze alte Position der Ghosts mit beliebigem Sprite
3. GameMap als Bild speichern

2 Umsetzung

Der Source Code wurde unterteilt in eine statische Library für das Unit Testing, welche die Ghost Logik beinhaltet, in ein Hauptprojekt für die Implementierung von z.B. File IO und dem Google Unit Test Projekt.

2.1 KI

Für die Ghosts und den Pacman wurde jeweils eine Klasse angelegt, welche jeweils die Position die Richtung und das Ziel gespeichert hat. Für jeden Ghost gibt es eine eigene Funktion, die das Ziel (Target) des Ghosts neu berechnet. Jeder Funktion wird der definierte Ghost und die entsprechend notwendigen Daten übergeben:

Listing 1: Prototypen für die Ghost Logik

```
void shadow_logic(Ghosts *shadow, Pac *pacman);
void speedy_logic(Ghosts *speedy, Pac *pacman);
void pokey_logic(Ghosts *pokey, Pac *pacman);
void bashful_logic(Ghosts *bashful, Ghosts *shadow, Pac *pacman);
```

Die Geistlogik wurde aus den gegebenen Unterlagen umgesetzt. Die Funktionen haben keinen Boardoverflow-Check. Dies bedeutet, dass beispielsweise ein Ghost auch ein Ziel außerhalb des Spielboards haben kann. Es wird mit diesen Daten jedoch nicht auf das GameBoard der Größe [28][31] zugegriffen und führt dadurch zu keinem falschen Boardzugriff.

Nachdem jeder Ghost ein Ziel erhalten hat, kann nun seine neue Richtung bestimmt werden. Um besser die originale Pacman KI zu verstehen, wurde folgende Quelle als Referenz für den implementierten Code verwendet: (Chad, 2010). Chad (2010) beschreibt, dass die Geister keine 180° Drehung durchführen dürfen, außer es gibt einen StateChange zu *escape*. Hier wird die Richtung der Geister um 180° gedreht bevor sie vor Pacman flüchten.

Somit ergibt sich folgende logische Implementierung der Berechnung der Bewegungsrichtung. In dem Source File *AI.cpp* werden der Funktion *ghostDirection* der zu berechnende Ghost übergeben, sowie die aktuelle GameMap (das gesamte "Spielbrett") und ein Flag (Setzen der neuen Position). Die Funktion überprüft nun alle möglichen Richtungen in der der Geist gehen kann (Up, Left, Down, Right). Stößt der Geist bei einer der Richtungen auf eine Wand, wird diese Richtung automatisch verworfen. Ist die Richtung möglich wird die Distanz zum Ziel des Ghosts mit der womöglich nächsten Position berechnet. Anschließend wird die "Reverse Direction" gesperrt, da eine Drehung um 180° verboten ist. Nun wird die kürzeste Distanz der zur Verfügung stehenden Richtungen berechnet und dem aktuellen Ghost übergeben. Mit dieser Logik wurde auch automatisch der Patrol-Modus gelöst. Es müssen keine Wegpunkte für diesen Modus definiert werden, es müssen nur die Home-Koordinaten als Ziel des Ghosts gesetzt werden und er patrouilliert automatisch im richtigen Bereich.

Nachdem jeder Ghost sein Ziel und seine gültige Bewegungsrichtung hat, können sie im Spielfeld bewegt werden. Dies geschieht in der Mainsource-File *PacMan_AI.cpp* in der Funktion *MoveGhosts()*. In der Funktion wird jeder Ghost aufgerufen und seine Position im Spielfeld gesetzt. Die alte Position wird mit einem Leerzeichen ersetzt. Die neue Position wird mit dem entsprechenden Char ersetzt, die dem Ghost und der aktuellen Richtung entspricht.

Bewegt sich ein Ghost in den Tunnel, gelangt er automatisch auf die andere Seite des Spielfeldes. In dieser Funktion wird auch überprüft, ob es zu einer Kollision von Geist und Pacman gibt → wenn ja, ist der Spieler GAME OVER.

3 Game Loop

Für eine schönere Darstellung wurde eine farbige Konsolenausgabe des Spielfeldes erstellt. Diese wird pro Spielzug einmal in der Konsole ausgegeben. Das Spielfeld wird in einen String geladen und zuerst der GlobalState überprüft. Anschließend wird ein zweidimensionales Char Array mit den Spieldaten befüllt und zeitgleich die Startposition von den Ghosts und Pacman gesetzt. Im Patrol-Modus wird jedem Geist sein Home-Corner als Ziel zugewiesen und die Logik der Geister deaktiviert. Im Escape-Modus wird zuerst die Richtung der Geister umgekehrt und anschließend das Ziel auf die negative Position von Pacman (zu der Logik im Escape-Modus der Geister wurde leider keine Information gefunden). Im Pursuit-Modus wird die KI ausgeführt. In allen Modi wird die Funktion zur Berechnung der neuen Richtung ausgeführt, da für diese Funktion lediglich das Ziel des jeweiligen Ghosts notwendig ist.

3.1 Grafische Ausgabe

Für die grafische Ausgabe mit den zur Verfügung gestellten Sprites wurde *SFML* (*Simple and Fast Multimedia Library* (<http://www.sfml-dev.org/>)) verwendet. In der Klasse Sprites werden die notwendigen Texturen und Spritenamen instanziiert und werden bei Programmausführung eingelesen. Anschließend wird das gesamte eingelesene Char-Array des Spielfeldes überprüft und zu dem jeweiligen gefundenen Char das zugehörige Sprite ausgegeben. Die Sprites wurden beim Einlesen in das Format $16 * 16$ skaliert. Somit kann das Sprite auf die Position des gefunden Char gesetzt werden → $PositionX/Y * 16(TileSize)$.

Mittels *window.capture()* wird ein Screenshot vom ersten Spielzug gemacht und in dem Verzeichnis der .exe Datei unter dem Namen "PacMan_Board_OneStep.png" gespeichert. Die Funktion für die Ausgabe wurde für Testzwecke so aufgebaut, dass sie die gesamte Gamelogik nochmals aufruft, sowie sich selbst. Damit kann das Spielverhalten der Ghosts besser visualisiert werden. Abbruchbedingung ist entweder ein GAME OVER → Ghost trifft Pacman oder das Fenster der Ausgabe wird geschlossen. Durch diese Funktionalität werden die Ghosts im Spielfeld automatisch bewegt und die Ausgabe in der Konsole, sowie in dem SFML-Window erfolgt.

In Abb. 1 sieht man ein Beispiel Input Spielfeld mit der dazugehörigen Ausgabe mit dem Tileset. Die Ghosts bewegen sich genau einen Spielzug. Die vorherige Position wird mit dem Ghost markiert, jedoch ohne Augen. Die Augen werden am Ghost der aktuellen Position mit der korrekten Richtung eingefügt. Sowohl der Ghost als auch Pacman hinterlassen danach ein leeres Feld. Die "dots" werden von allen Charakteren gegessen. Im Escape-Modus werden die Geister mit dem entsprechenden Sprite ersetzt.

```
GlobalState=pursuit
5111111111111111ab11111111111116
4..J.....DB.....2
4.EAAF.EAAF.DB.EAAF.EAAF.2
4*D B.D B.DB.D B.D B*2
4.GCCH.GCCCH.GH.GCCCH.GCCH.2
4.....n.....2
4.EAAF.EF.EAAAAAAF.EF.EAAF.2
4.GCCH.DB.GCCFECCH.DB.GCCH.2
4.....DB....DB....DB.....2
833333).DGAAF DB EAAHB.(33337
    4.DECCH GH GCCFB.2
    4.DB          DB.2
    4.DB (3>--<3) DB.2
11111].GH 2      4 GH.[11111
R . 2      4 .
33333).EF 2      4 EF.(33333
    4.DB [11111] DB.2
    4.DB          DB.2
    4.DB EAAAAAAF DB.2
51111].GH GCCFECCH GH.[11116
4.....DBO.....2
4.EAAF.EAAF.DB.EAAF.EAAF.2
4.GCFB.GCCCH.GH.GCCCH.DECH.2
4*..DB.....1...DB..*2
gAF.DB.EF.EAAAAAAF.EF.DB.EAc
hCH.GH.DB.GCCFECCH.DB.GH.GCd
4.....DB....DB....DB.....2
4.EAAAAHGAAF.DB.EAAHGAAAAAF.2
4.GCCCCCCCCH.GH.GCCCCCCCCH.2
4.....2
83333333333333333333333333337]
```

(a) Input Spielfeld



(b) Output Game Map mittels SMFL

Abbildung 1: Subfigure Caption

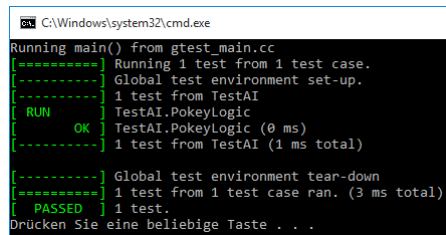
4 Google Unit Testing

Google Unit Testing ist verfügbar von <https://github.com/google/googletest>. Der Source Code wurde in eine statische Library kompiliert und ein neues Projekt für den Test angelegt. Um die KI im Google Framework zu testen, wurde der notwendige Code ebenfalls in eine statische Library kompiliert und in das Projekt, sowie das Hauptprojekt, eingebunden. Das Projekt *unitTest_Pacman* hat nun Zugriff auf die verwendeten Klassen und Objekte. Das Hauptprogramm kann in diesem Testszenario weiterhin ausgeführt werden. Für den Test werden die Klassen für die Ghosts und den Pacman eingebunden, sowie die implementierte KI. Nun kann ein simples Testszenario erstellt werden, indem die Logik für den Ghost Pokey getestet wird. In dem Test werden zwei neue Objekte instanziiert und auf einen vordefinierten Wert gesetzt. Pokey hat eine zweifache Logik. Ist Pokey in der Nähe von Pacman (8 Felder) dann wird er zurück in seinen Home-Corner geschickt. Wenn er jedoch weiter entfernt ist, wird das Ziel auf die exakte Position von Pacman gesetzt. Der Test überprüft beide Szenarien, indem die Position von Pokey verändert wird:

Listing 2: Google Unit Test Beispiel; Shadow Logik

```
TEST(TestAI, PokeyLogic)
{
    Ghosts pokeytest;
    Pac pactest;
    pactest.setpos(10, 1);
    pactest.setdir(1, 0);
    // set pokey close to pacman -> go home
    pokeytest.setpos(8, 1);
    pokey_logic(&pokeytest, &pactest);
    EXPECT_EQ(pokeytest.targetx, 1);
    EXPECT_EQ(pokeytest.targety, 29);
    // set pokey far from pacman -> go to pac pos
    pokeytest.setpos(20, 1);
    pokey_logic(&pokeytest, &pactest);
    EXPECT_EQ(pokeytest.targetx, pactest.getposx());
    EXPECT_EQ(pokeytest.targety, pactest.getposy());
}
```

Die Werte sind aus der Logik her einfach zu setzen und der Google Unit Test liefert folgendes Ergebnis:



```
C:\Windows\system32\cmd.exe
Running main() from gtest_main.cc
[*****] Running 1 test from 1 test case.
[*****] Global test environment set-up.
[*****] 1 test from TestAI
[ RUN   ] TestAI.PokeyLogic
[ OK    ] TestAI.PokeyLogic (0 ms)
[*****] 1 test from TestAI (1 ms total)

[*****] Global test environment tear-down
[*****] 1 test from 1 test case ran. (3 ms total)
[ PASSED ] 1 test.
Drücken Sie eine beliebige Taste . . .
```

Abbildung 2: Google Unit Test Ergebnis → Pokey Logik

5 Programmausführung

In dem Abgabepfad im Verzeichnis *bin* befinden sich zwei .EXE Files (x64). **PacMan_AI.exe** ist das Hauptprogramm, welches die gesamte Funktionalität beinhaltet. Das Programm zeigt in der Konsole das aktuelle Spielfeld in Farben an. Weiters wird ein SFML Window geöffnet, welches den gleichen Spielstand mit den Sprites/Tiles anzeigt. Folgende Möglichkeiten stehen zur Verfügung um die .EXE Datei auszuführen:

1. PacMan_AI.exe ausführen: Hierzu muss das gegebene Spielfeld in dem Ordner der .EXE mit dem Namen: *< TestMap.txt >* existieren. In der Abgabe ist dieses File bereitgestellt und in dem Ordner bereits angelegt.
2. PacMan_AI.exe mittels Command Line ausführen: Command Line in dem Ordner der .EXE starten. PacMan_AI.exe mit dem Input Argument `--load "Input File"` starten: *PacMan_AI.exe --load TestMap.txt*

Das zweite .EXE File **unitTest_Pacman.exe** führt Unit Tests für alle Ghosts aus. Das Programm sollte mittels der Command Line ausgeführt werden: *unitTest_Pacman.exe* liefert anschließend die Ergebnisse der Tests in der Konsole (siehe Beispiel Abb. 2).

Die Tiles und Sprites befinden sich in dem Unterverzeichnis *pacman*. Die notwendigen DLLs für SFML und die Library für den Google Test sind inkludiert.

6 NICE-TO-HAVE

Folgende Punkte wurden implementiert:

1. Farbausgabe in der Konsole: Das Spielfeld wird nach jedem Spielzug in der Konsole farbig ausgegeben.
2. Richtungsindikator für Augen: Je nach aktueller Richtung wird dem Ghost das korrekte Augen-Sprite eingefügt. An Kreuzungen und Ecken wird das Augen-Sprite der nächsten Richtung eingefügt (rein visuell hat dies für den Spielfluss einen schöneren Eindruck gemacht).
3. KI für Pacman: Es wurde eine sehr simple KI für Pacman implementiert. Hauptsächlich dient diese Implementierung dazu, Pacman durch das Spielfeld zu führen. Pacman berechnet sich die Distanz zu den Ghosts und speichert sich den Ghost der am nächsten ist. Das Ziel von Pacman wird auf einen Zufallswert in einem definierten Bereich gesetzt und die Position des berechneten Ghosts wird subtrahiert. Pacman darf, wie die Ghosts, keine 180° Drehung in dieser Implementierung durchführen.

Literatur

Chad, B. (2010). Understanding Pac-Man Ghost Behavior. *GameInternals*. Verfügbar von: <http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior> [Zugriff am 2015-10-20].