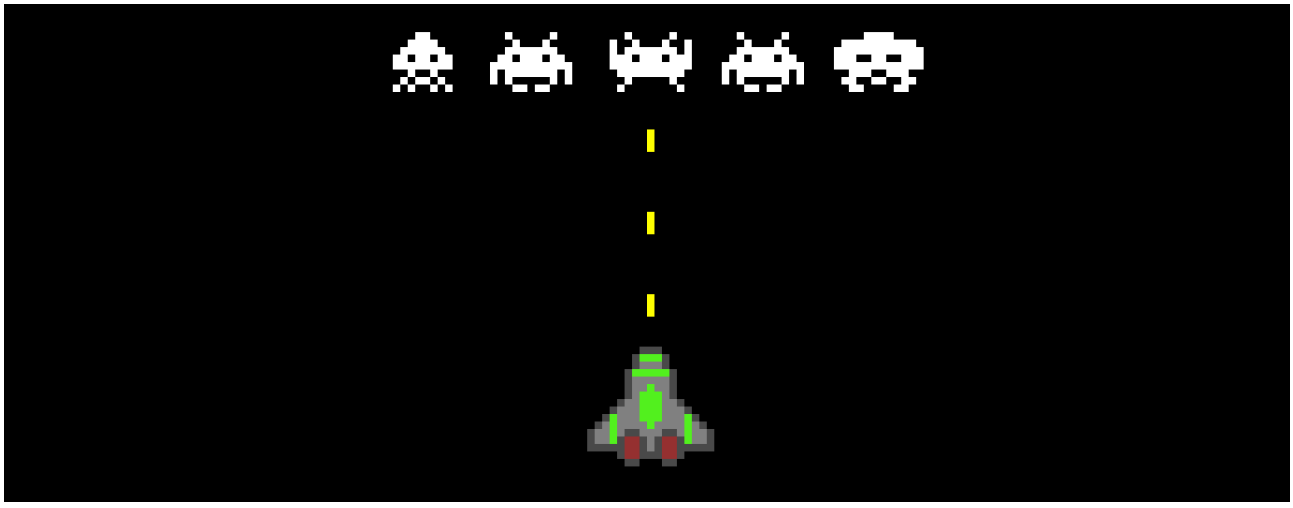
Heinrich Heine Universität Düsseldorf

Hardwarenahe Programmierung - Abschlussprojekt

- Die Aufgabe ist erstmals nicht kleinschrittig für Sie vorbereitet. Lesen Sie zuerst die ganze Aufgabenstellung. Fertigen Sie danach einen grobe Skizze in Form von Kommentaren oder Funktionsaufrufen an. Ersetzen Sie die Kommentare dann durch funktionsfähigen Code. Schreiben Sie außerdem eigene Tests und führen Sie diese oft aus.
- Sie müssen das Abschlussprojekt in einem Einzelgespräch erklären und dazu Fragen beantworten. Die Terminbuchung für die Einzelgespräche *zur Präsentation* müssen über das ILIAS gebucht werden. Suchen Sie dort nach "Terminbuchung Präsentation Abschlussprojekt".
- Tests können nur beweisen, dass ein Programm Fehler hat. Das Gegenteil zu zeigen, also das ein Programm immer fehlerfrei arbeitet, ist nahezu unmöglich. Verlassen Sie sich daher nicht zu sehr auf unsere Tests, sondern testen Sie selbst die Sonderfälle in Ihrer Implementierung.
- Testen Sie **vor** der Präsentation Ihre technische Ausstattung (Webcam/Mikrofon oder Smartphone) und halten Sie einen **Lichtbildausweis** bereit. Auf dem Studierendenausweis ist kein Bild von Ihnen, sondern von Heinrich Heine.
- Termine für Einzelgespräche bei Problemen mit der Aufgabe (*nicht zur Präsentation*):
  - Mittwoch, der 10. November, von 12:30 Uhr bis 16:00 Uhr.
  - Donnerstag, der 11. November, von 10:30 Uhr bis 14:00 Uhr.
  - Freitag, der 12. November, von 8:30 Uhr bis 12:00 Uhr.
  - Mittwoch, der 17. November, von 12:30 Uhr bis 16:00 Uhr.
  - Donnerstag, der 18. November, von 10:30 Uhr bis 14:00 Uhr.
  - Freitag, der 19. November, von 8:30 Uhr bis 12:00 Uhr.

1 Abschlussaufgabe – Space Invaders



Bei diesem Projekt werden Sie eine Variation des Spiels "Space Invaders" implementieren.

In dieser Aufgabenstellung sind viele Details absichtlich nicht genauer ausgeführt, damit Sie ihrer Kreativität freien lauf lassen können. Fragen Sie die Tutoren oder Tutorinnen, wenn Sie sich unsicher sind, ob Sie etwas Bestimmtes machen dürfen oder nicht.

2 Aufgabenstellung:

Implementieren Sie eine Variante des Spiels "Space Invaders" in den Dateien `spaceinvaders.c` und `spaceinvaders.h`. Sie müssen dabei nicht das Original bis in das letzte Detail nachbauen, sondern können davon abweichen, wenn das Spiel dann mehr Spaß machen sollte. Es müssen aber mindestens die folgenden Spielelemente implementiert werden:

1. Das Spiel muss auf der Standardausgabe dargestellt werden.
2. Es muss ein Raumschiff geben, das über die Standardeingabe gesteuert werden kann.
3. Es muss gegnerische "Raumschiffe" (Gegner) geben.
4. Es muss einen "Endboss" geben, der mindestens 3 mal so groß wie die normalen Gegner ist.
5. Das Raumschiff muss schießen können und die Geschosse müssen die Gegner zerstören können.
6. Die Gegner wandern vom linken zum rechten Spielfeldrand und wieder zurück.
7. Die Gegner wandern nach unten, aber deutlich langsamer, als sie nach links und rechts wandern.
8. Wenn sich Gegner und Raumschiff berühren, dann nehmen beide Schaden.
9. Das Spiel ist verloren und wird beendet, wenn das Raumschiff zerstört wird.
10. Das Spiel ist verloren und wird beendet, wenn ein Gegner den unteren Rand erreicht.
11. Das Spiel ist gewonnen und wird beendet, wenn alle Gegner und der Endboss zerstört wurden.
12. Um den Endboss zu zerstören sind mehrere Schüsse nötig.
13. Der Endboss muss aktiv versuchen, das Raumschiff zu zerstören, darf also nicht einfach nur herumsitzen und nichts tun.
14. Es müssen verschiedene Level geladen werden können.
15. Es muss verschiedene Schwierigkeitsgrade geben.
16. Sie müssen mindestens ein `struct` und ein `enum` deklarieren und *sinnvoll* verwenden.
17. Das Spiel muss durch die Eingabe des Zeichens "q" beendet werden können. Sie dürfen annehmen, dass danach die Enter-Taste gedrückt wird.

Zusätzlich müssen Sie die eigene Tests schreiben, Fragen in der Datei `Fragen.txt` beantworten und einen Screenshot vom Spiel machen, in dem der Endboss zu sehen ist. Die Datei mit dem Screenshot darf nicht größer als 500 kB sein und muss `endboss.png` heißen.

Zur Unterhaltung können Sie mit den folgenden Befehlen unter Ubuntu ein ähnliches Spiel installieren:

```
sudo apt install ninvaders
```

Das Spiel lässt sich dann mit dem Befehl `ninvaders` starten.

3 Kommandozeilenoptionen

Das Programm muss die folgenden (optionalen) Kommandozeilenoptionen verarbeiten können:

- `--level <filename>` soll das Level mit dem gegebenen Dateinamen laden.
  - Wenn diese Option nicht übergeben wurde soll das Level `level/1.txt` geladen werden.
- `--difficulty <value>` soll den Schwierigkeitsgrad setzen.
  - Gültige Werte sind: `easy`, `normal` (Standardwert) und `hard`.

Beispiele:

- Startet das Spiel mit Schwierigkeitsgrad `normal` und Level `level/1.txt`.

  ```
  ./spaceinvaders
  ```

- Startet das Spiel mit Schwierigkeitsgrad `easy` und Level `level/1.txt`.

  ```
  ./spaceinvaders --difficulty easy
  ```

- Startet das Spiel mit Schwierigkeitsgrad `normal` und Level `level/2.txt`.

  ```
  ./spaceinvaders --level level/2.txt
  ```

- Startet das Spiel mit Schwierigkeitsgrad `hard` und Level `level/3.txt`.

  ```
  ./spaceinvaders --level level/3.txt --difficulty hard
  ```

4 Level

Um das Spiel interessanter zu machen soll es mehrere Level geben. Das Level-Format ist Ihnen überlassen. Erstellen Sie mindestens die fünf folgenden Level:

- `level/1.txt` – Dieses Level ist das Standardlevel.
- `level/2.txt` – Ein weiteres Level.
- `level/3.txt` – Ein Level, das sehr breit ist (mindestens 10000 Zeichen)
- `level/4.txt` – Ein Level, das sehr hoch ist (mindestens 10000 Zeilen)
- `level/5.txt` – Ein Level, das mindestens 10000 Gegner enthält.

5 Rückgabewerte

Das Programm soll im Fehlerfall eine aussagekräftige Fehlermeldung auf der Standardfehlerausgabe ausgeben und folgende Rückgabewerte liefern:

- 0, wenn das Spiel ordnungsgemäß beendet wurde (verloren, gewonnen, beendet mit “q”).
- 1, wenn falsche Kommandozeilenparameter übergeben wurden, ungültige Dateien ausgenommen.
- 2, wenn die Level-Datei nicht geöffnet werden konnte oder nicht existiert.
- 3, wenn die Level-Datei nicht gelesen werden konnte. Lesefehler können Sie am Rückgabewert Ihrer Einlesefunktion erkennen. Lesen Sie dazu die Dokumentation. Ob ein Lesezugriff aufgrund einer leeren Datei oder einer wegen eines “richtigen” Fehlers fehlschlug können Sie *nach* dem Leseversuch mit der Funktion `ferror` prüfen.
- 4, wenn die Level-Datei leer ist.

Falls gleichzeitig verschiedene Fehler auftreten sollten darf Ihr Programm sich einen davon aussuchen, diesen wie oben behandeln und den entsprechenden Rückgabewert liefern.

6 (Optionale) Echtzeiteingabe

Damit das Spiel mehr Spaß macht wäre es schön, wenn das Spiel nicht nach jeder Eingabe darauf warten würde, dass die Enter-Taste gedrückt wird. Leider ist das Terminal "line buffered", was bedeutet, dass man die Eingabe eigentlich nur Zeilenweise verarbeiten kann, wenn man sich an den C99-Standard hält.

Wir haben Ihnen in der Datei `keyboard.c` die Funktion `int wait_for_key(int milliseconds)` zur Verfügung gestellt, die eine gewisse Anzahl von Millisekunden wartet und als Rückgabewert ein in dieser Zeit eingegebenes Zeichen liefert, selbst wenn kein Zeilenumbruch erfolgt ist. Ausnahmsweise dürfen Sie diese Funktion benutzen, obwohl sie plattformspezifische Erweiterungen verwendet.

Alternativ können Sie die Eingabe aber auch wie gewohnt von der Standardeingabe einlesen und das Spiel durch Drücken der Enter-Taste vorwärtstreiben, falls die Funktion `wait_for_key` auf Ihrem Betriebssystem nicht funktionieren sollte oder Sie sie nicht verwenden möchten.

Weiterhin ist noch die Funktion `void clear(void)` gegeben, die das Terminal löscht, wenn Ihr Terminal diese Funktionalität unterstützt. Auch diese Funktion müssen Sie nicht verwenden.

7 Testen des Programms

Wir stellen Ihnen eine Makefile-Datei zur Verfügung. Mit dem Befehl `make run` können Sie ihr Programm kompilieren und testen. Dadurch werden mehrere Unterregeln ausgeführt:

- `make spaceinvaders_valgrind`: Kompiliert das Programm ohne `fsanitize`, damit es mit dem Programm `valgrind` ausgeführt werden kann.
- `make spaceinvaders_fsanitize`: Kompiliert das Programm mit den zusätzlichen Compiler-Optionen `-fsanitize=address` und `-fsanitize=undefined`.
- `make argests`: Überprüft diverse Kommandozeilenparameter.
- `make codestyle`: Überprüft die Code-Vorgaben.
- `make questions`: Prüft, ob alle Fragen beantwortet wurden.
- `make clean`: Löscht Objekt-Dateien und kompilierte Programme.
- `make mytests`: Soll die Tests ausführen, die Sie selber erstellt haben.

8 Eigene Tests

Da wir Ihnen bei dieser Aufgabe sehr viel Freiheit lassen, können wir Ihnen keine Tests zur Verfügung stellen, die den Spielablauf testen. Schreiben Sie hierzu eigene Tests. Erstellen Sie hierzu im `Makefile` die Regel `mytests`, die Ihre Tests kompiliert und ausführt. Sie dürfen auch weitere Regeln erstellen. Ob Sie die Tests mit `check`, Shell-Skripten oder eigenen Programmen realisieren bleibt Ihnen überlassen. Sie sollten mit den Tests möglichst viele Möglichkeiten für Speicherzugriffsfehler ausschließen. Beispielsweise könnten Sie prüfen, ob der Endboss nicht unter ungünstigen Umständen die Level-Grenzen überschreiten könnte, um auf ungültige Speicherbereiche zuzugreifen.

Denken Sie daran, dass Sie in Ihren Tests sowohl `fsanitize` als auch `valgrind` nutzen.

9 Wichtige Anforderungen:

- Der Speicher ihrer Datenstrukturen muss mit `malloc` und `free` dynamisch verwaltet werden. Insbesondere darf es keine feste Obergrenze für die Größe des Levels oder die Anzahl an Gegnern geben, ab der Ihr Programm plötzlich nicht mehr funktioniert. Dies bedeutet auch, dass Sie das Spielfeld und Gegner nicht auf dem Stack anlegen dürfen, denn der Stack hat eine begrenze Größe. Die Funktionen `calloc` und `realloc` dürfen auch verwendet werden.

- Alle Arrays mit mehr als 100 Elementen müssen dynamisch alloziert werden. Um die Wahrscheinlichkeit zu erhöhen, dass diese Regel nicht ignoriert wird, ist die Verwendung von Konstanten größer als 100 nicht erlaubt.
- Vor jeder Beendigung des Programms muss der gesamte Speicher wieder freigegeben werden und geöffnete Dateien müssen geschlossen werden.
- Speicherzugriffsverletzungen, Speicherlecks oder sonstigen Fehlern dürfen unter keinen Umständen auftreten.
- `valgrind` und `fsanitize` dürfen für alle erreichbaren Spielzustände keine Fehler melden.
- Funktionen sowie Datenstrukturen müssen in einer separaten Header-Datei deklariert werden.
- Das Programm muss alle Tests fehlerfrei bestehen, d.h. `make run` darf keine Fehler liefern.
- Ein Testdurchlauf (`make run`) darf nicht länger als fünf Minuten dauern und nicht mehr als 1 Gigabyte RAM belegen.
- Sie müssen Ihren Code in einem Einzelgespräch erklären und Fragen beantworten. Reservieren Sie dazu im ILIAS einen Termin. Geben Sie ihren Termin bitte rechtzeitig ab, wenn absehbar ist, dass die Tests zum Ende der Abgabefrist nicht erfolgreich durchlaufen oder sonstige Anforderungen nicht erfüllt werden.

Viel Erfolg!