Project Structuur van Computerprogrammas 2

Alexandre Kahn
alexandre.kahn@vub.be
500067
BA Computerwetenschappen
Faculteit Wetenschappen en Bio-ingenieurswetenschappen

14 januari 2017

Inhoudsopgave

1 Inleiding

Voor dit project moesten we onze eigen versie van Snake implementeren. Een deel van mijn code heb ik gebaseerd op de taak waar we Minesweeper moesten schrijven. Zo is leunt een deel van de implementatie op de grid die we voor minesweeper hadden, dit maakt het gemakkelijk om met absolute coördinaten te werken en de appel op de nodige plekken te laten verschijnen. De gegevens worden weggeschreven naar .txt bestanden, net zoals in mijnenveger ook al moest gebeuren. Ook is de grafische interface en de user input gelijkaardig aan de vorige taak.

2 Design

2.1 Grid

De grid is een struct die verschillende staten kan hebben. Zo kan deze een `APPLE`, `SPECIAL_APPLE`, `WALL` of `NORMAL` zijn. De implementatie is gebaseerd op die van de taak, dit omdat het zo zeer overzichtelijk was waar alles zich bevond en ik zo gebruik maak van abstracte coördinaten. De appel wordt steeds, nadat deze opgegeten is opnieuw op een willekeurige plaats gegenereerd, terwijl de speciale appel dezelfde code gebruikt om ook aangemaakt te worden. De muren worden ook in de grid geplaatst, deze worden uit een bestand geladen en dan correct aangepast. Door aanpassingen in dat bestand te maken is het mogelijk om de muren aan te passen.

2.2 Snake

De slang is na lang zoeken een gelinkte lijst geworden. Eerst was mijn plan hier een array van te maken, en deze elke keer dat de slang groeit te heralloceren. Het probleem was echter dat de `realloc` functie het geheugen niet bijhield. Zo gebeurde het dat de geheugen adressen verplaatst werden en dat men de inhoud van deze cellen verloor. Een oplossing was om steeds de geheugeninhoud te overschrijven, maar dit was niet echt efficiënt. Bij gelinkte lijsten hebben we dit probleem niet. De slang wordt elke keer dat hij groeit langer. Bij het afsluiten van het spel wordt het gebruikte geheugen ook vrij gegeven. Dit is belangrijk om memory leaks te voorkomen.
Een laatste toevoeging is het mogelijk te maken om met 2 te spelen. Deze code had zeker wat mooier kunnen zijn, mits ik meer tijd had om het project af te maken. Het werkt door bij op de W toets (op een qwerty toetsenbord) te klikken. Dan wordt de 2e slang gealloceerd en opgestart. Ook begint deze direct te bewegen en reageert deze op de input van de WASD-toetsen.

2.3 GUI

De GUI initialiseert bij het opstarten al de grafische zaken. Zo zal het venster met het speelveld, de slang, muren en een appel verschijnen. De score wordt met de `SDL_TTF` library ook weergegeven. Dit is momenteel in de linkerbovenhoek waar het niet teveel stoort, maar zou in een verder uitgewerkte implementatie bijvoorbeeld in een extra balk aan de zijkant kunnen staan.
SDL wordt ook gebruikt voor de user input. De functie `read_input` doet dit. Hier had ik enkele problemen met de switch case structuur dus heb ik bepaalde zaken door een if, else if veranderd. Zo krijg ik geen problemen dat een toetsaanslag voor meerdere zaken telt.
Elke speltick wordt het speelveld opnieuw getekend. Dit begint met eerst elke cel, die een vakje, een appel of een muur kan zijn. Daarboven wordt de slang getekend.

2.4 I/O

Hieronder vallen verschillende functies. Zo wordt de highscore steeds geladen indien deze file al bestaat. Deze wordt op het einde van elk spel overschreven indien dit nodig is. Aan het begin van elk level wordt de staat van de muren ingeladen, hier zorgt de `load_walls` functie voor. Bij het opstarten wordt er tevens gezien of er nog een een opgeslagen spel is. Indien dit het geval wordt dat geladen in plaats van een nieuw spel geïnitialiseerd, deze files wordt dan verwijderd door diezelfde functie.
Als het spel wordt afgesloten zonder dat de speler dood is wordt alles ook netjes in een bestand opgeslagen. Zowel de locatie van de appel als de locatie en lengte van de slang worden dan opgeslagen.

2.5 Gameloop

De gameloop wordt door de main opgeroepen en blijft runnen zolang er niet op P (voor pauze) is geklikt of dat de speler dood is. In deze functie wordt er ook op bepaalde tijdsintervallen de speciale appel geactiveerd. Deze zal dan voor een aantal seconden op het scherm blijven, alvorens de functie deze weghaalt indien deze niet opgegeten was.
Er is ook een functie om het spel te pauzeren, deze zet de staat van het spel omgekeerd aan wat deze hiervoor was.

3 Testing

Het spel is zeer clichématig getest door het te spelen. Dit is een van de beste manieren volgens mij om te zien dat het in een normaal geval correct verloopt. Verder heb ik ook steeds in implementatiefases getest, zo ging ik steeds na of het nieuwe stuk naar behoren werkte voor ik er een ander stuk bij deed.
Voor het spel op te slaan heb ik eerst na gegaan of de output werkte, dit door screenshots

te nemen van het speelveld en dan na te gaan of alles klopte. Toen de functies voor het importeren ook werkten heb ik dan natuurlijk getest of alles werkte, door ook zelf zaken in de file aan te passen en te zien hoe het spel daarop reageerde.
Tot slot heb ik naar potentiële memory leaks gezocht. Hiervoor heb ik het programma Instruments dat een onderdeel van Xcode is gebruikt, Valgrind kreeg ik jammer genoeg niet werkende op de nieuwste versie van MacOS.

4 Code

4.1 Main.c

Bevat:
`int main(int argc, char *argv[])`

4.2 input_output.c

Bevat:
```
void open_file()
void close_file()
void save_to_file()
void update_highscore(int score)
void save_snake_state(int grid_height, int grid_width)
void save_apples_state(int grid_height, int grid_width
void load_snake_state(int grid_height, int grid_width)
void load_apple(int grid_height, int grid_width)
void load_walls(int grid_height, int grid_width)
void change_direction(int direction, int snake_nr)
struct Snake * get_snake(int nr)
int get_nr_of_snakes()
```

4.3 game.c

Bevat: `void game_loop(int width, int height)`
`void pause_game()`

4.4 snake.c

Bevat:
```
int get_score()
void set_score(int new)
void allocate_snake(int height, int width, int snake_nr)
```

```
void initialize_snake(int width, int height, int snake_nr
void deallocate_snake(snake_nr)
void extend_snake(int x, int y, int snake_nr
void snake_eat(int x, int y, int snake_nr)
void move_head(int width, int height, int snake_nr)
void check_apple(int width, int height, int x, int y, int snake_nr)
void check_wall()
bool check_bodyparts_loop(int x, int y, int snake_nr)
void check_bodyparts_collision()
void move_tail(int snake_nr)
void move_snakes(int width, int height)
```

4.5 gui.c

Bevat:

```
void drawText(SDL_Surface* screen, char* string, int size, int x, int y)
void stop_gui()
void draw_cell(int x, int y, int kleur)
void draw_snake_part(int part)
void draw_snake_head()
void draw_grid(int width, int height)
void draw_game_over(int width, int height)
void read_input(int width, int height, bool game_running)
void initialize_figures()
void initialize_window(char *title, int grid_width, int grid_height)
void initialize_gui(int grid_width, int grid_height)
```

4.6 grid.c

Bevat:

```
struct Cell *** allocate_grid(int grid_width, int grid_height)
static struct Coordinate* generate_random_apple(int grid_width, int grid_height)
static struct Coordinate* generate_walls(int grid_width, int grid_height)
void place_apple(struct Coordinate apple_coordinates[], int grid_width, int grid_height)
void place_walls(struct Coordinate wall_coordinates[], int nr_of_walls)
void make_walls(int grid_height, int grid_width)
void make_apple(int grid_height, int grid_width)
void initialize_grid(int grid_height, int grid_width)
```