SNEK

API Documentation and Project Guideline

ACCESSING THE API

You can access the API by cloning the git repository using the same SSH credentials that you use for the labs. Navigate to your directory of choice using `cd`, then:

```
git clone ssh://gitolite/projects/
```

Run `ls` to see the `snek` directory. Using the API is **optional** but highly recommended.

OPERATIONS

The following operations should be reflected in the interim and final reports:

1. Augment the API such that it can accept a sequence of moves in a data structure of your choice and execute these moves in the game.
2. If you are working with the API in C, ensure that your program **runs Valgrind clean**.
3. Design and implement an algorithm to maximize the points Magini acquires before the game ends using the default `BOARD_SIZE` (see Constants section below) of 10.
4. Run your algorithm over 100+ trials to demonstrate its efficacy (automate this process using e.g. Python or a bash script). Store the final scores across trials as a dataset (you may pipe the output to a text file) and quantify the variance of its performance. What is the expected score for your algorithm? What is the minimum score your algorithm is guaranteed to accomplish?
5. Choose at least one of the following augmentations to implement:
    a. Extend the capability of your algorithm to work for varying board sizes. Find a lower and upper bound on the `BOARD_SIZE` that your algorithm can reliably work on. Demonstrate this quantitatively.
    b. Identify the smallest `CYCLE_ALLOWANCE` (see Constants sections below) your algorithm works for. Demonstrate this quantitatively.
    c. Run 50+ trials of your algorithm across `BOARD_SIZEs` and/or `CYCLE_ALLOWANCEs`. Identify the relationship between your algorithm's performance and these parameters.

API OVERVIEW

Your name is Voldy. You have a (bleppy) pet snake (snek) named Magini, and you both want to play a game with the Moogles. The game starts off in a playing field, where Magini occupies a single cell at $(0,0)$ and has a length of 1 block. The API has been developed with a rudimentary graphical interface for you to use, shown in figure 1.

The playing field is a coordinate system where the top-most left-most cell has coordinate $(0,0)$. You may find the convention used for this API in figure 2.

Magini has a sort of inertia; until commanded to change direction, she will slither onwards, undisturbed. Magini can move in one of four directions: {UP, DOWN, RIGHT, LEFT}, and requires an additional specification of the axis {AXIS_X, AXIS_Y}. You can command Magini once per change in cell. As soon as Magini moves a single cell, the field refreshes and the chance that there is a <TARGET> arises.

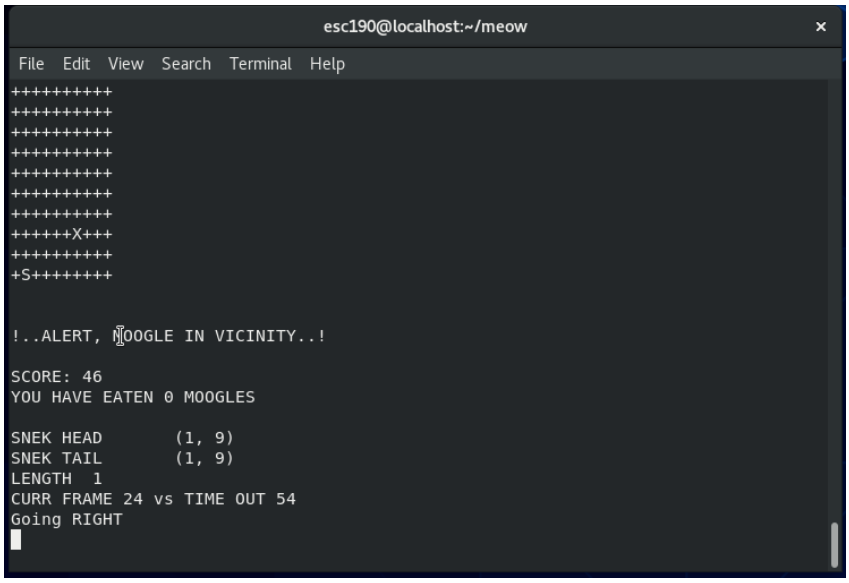

Figure 1: The snek API playing field.

If a <TARGET> appears (a Moogle or Hurry Pooter), the cell contents will contain the score. **Only one <TARGET> will appear on the playing field at a time**. Once Magini eats the target, the cell resets to a score of zero, the user's score increments by that score, and Magini's length increments by 1. Magini gets angry when she: (1) bumps into an edge on the playing field or (2) bumps into herself, and the game ends. The game also ends when a Moogle appears and Magini does not eat the Moogle before a set `TIMEOUT`. Your task is to create an algorithm to maximize the points you get before the game ends.

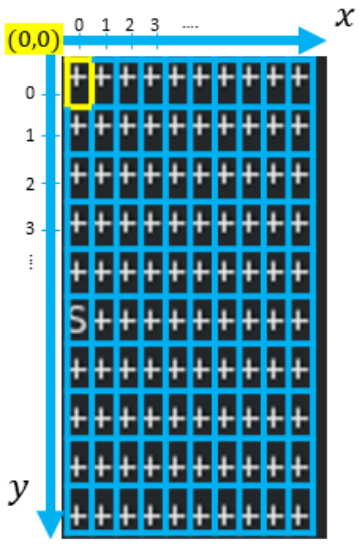

Figure 2: Snek coordinate system. Magini's head is at $(x,\ y) = (0, 5)$.

API USAGE (C)

You can first run the provided executable `./snek_AI` to see the interface shown in figure 1. The Snek API has been provided as a pre-compiled binary, `snek_api.o`. If you choose to work in C in `main.c`, you will need to first compile your `.c` file without linking, then link the two compiled binaries as shown below:

```
gcc main.c -o main.o -c
gcc -o <TARGET OUTPUT FILE> main.o snek_api.o
```

The `gcc` flags indicate:

```
-c: stop at the compiling stage, i.e. do not attempt to link
-o: target output file
```

You may read the full documentation on gcc using the following terminal command:

```
man gcc
```

Lastly, you may find it helpful to use a `makefile` to speed up the compiling and linking process. A sample makefile has been provided for your reference. Try out the following commands:

```
make clean
make snek_AI
```

API USAGE (PYTHON)

To access the adapted API for Python, navigate to the `py` subdirectory. A sample program has been created for you which you can run using:

```
python3 main.py
```

The C API was ported into Python using the `ctypes` library and is found in `snek.py`. Note that if you make changes to e.g. `BOARD_SIZE` in the Python file, you **must** modify `snek_api.h` to reflect this change and recompile using:

```
gcc -fPIC -shared -o libsnek_py.so snek_api.c
```

Or alternatively, run (from the `snek` directory):

```
make libsnek_py.so
```

The code in `main.py` indicates usage of the functions provided by the API. They are functionally equivalent to the description in the Function Declaration section. Note two useful functions from the `ctypes` library:

`byref()`: reference the argument; i.e. pass by reference

`POINTER()`: creates a pointer-type object for Python to recognize

Additionally, to dereference a pointer; e.g. if `ptr` is a pointer to a `struct`, the struct member `member` can be accessed via

```
ptr[0].member
```

CONSTANTS

The header file `snek_api.h` contains constant definitions that you **may wish to change** for testing purposes.

`BOARD_SIZE`

Determines the size of the game board.

`CYCLE_ALLOWANCE`

Determines the `TIMEOUT` as related to the perimeter of the board. `TIMEOUT` sets the number of frames which can elapse **after** the appearance of a Moogle (i.e. without eating it) before the game ends. In other words, as soon as a Moogle appears, you must eat that Moogle before `TIMEOUT` number of frames elapse otherwise the game will end. You can adjust `CYCLE_ALLOWANCE` to be more lenient for testing. The default is 1.5. The timeout is computed as:

```
int TIMEOUT = (int)((4 * BOARD_SIZE - 4) * CYCLE_ALLOWANCE)
```

STRUCTS

The header file `snek_api.h` contains struct declarations which you **should not change**.

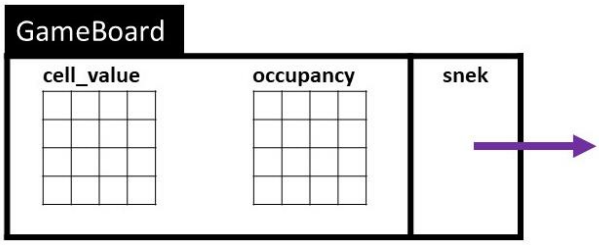

Figure 3: Visual representation of the GameBoard struct.

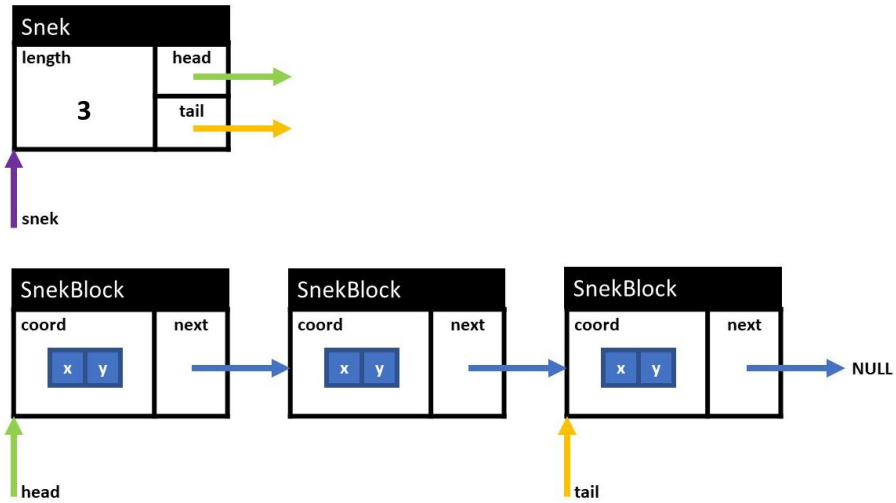

Figure 4: The usage of Snek and SnekBlock structs.

`GameBoard:` Contains information about the playing field. See figure 3 for a visual representation.

`int cell_value[BOARD_SIZE][BOARD_SIZE]`

Array of size `BOARD_SIZE x BOARD_SIZE`. Keeps track of whether a <TARGET> is on the playing field. A <TARGET> is one of {Moogle, Hurry Pooter}. **Only one <TARGET> will appear on the playing field at a time**. If there is no <TARGET> at coordinate (x, y), `cell_value[y][x]` has a value of 0. If a regular Moogle occupies index (x, y), `cell_value[y][x]` has a value of 20. If Hurry Pooter occupies index (x, y), `cell_value[y][x]` has a value of 60.

`int occupancy[BOARD_SIZE][BOARD_SIZE]`

Array of size `BOARD_SIZE x BOARD_SIZE`. Keeps track of all blocks on the playing field that the Snek occupies. `occupancy[y][x]` is 0 if the Snek does not occupy coordinate (x, y) and is 1 if Snek occupies coordinate (x, y).

`struct Snek* snek`

Pointer to Snek. See the section on `Snek`.

`Snek:` Keeps track of a Snek on the playing field. Created as a member of a `GameBoard` struct. Refer to `SnekBlock` (implemented as a linked list). See figure 4 for a visual representation.

`struct SnekBlock *head`

Pointer to SnekBlock which is the head of the Snek.

`struct SnekBlock *tail`

Pointer to SnekBlock which is the tail of the Snek.

`int length`

The length of the `Snek`; i.e. how many `SnekBlocks` it is composed of. The starting length is 1.

`SnekBlock`: A single block of the Snek.

`int coord[2]`

The coordinate (x, y) of the SnekBlock.

`struct SnekBlock *next`

Pointer to the next SnekBlock.

FUNCTION DECLARATIONS

The header file `snek_api.h` contains function declarations which you **should not change**. The descriptions for the functions which you will likely use are indicated below. You may submit a request for descriptions of other function declarations found in `snek_api.h` via Discourse.

`GameBoard *init_board()`

> Initializes a square game board with size `BOARD_SIZE` and a new `Snek` at $(0,0)$. Returns a pointer to the `GameBoard` struct. The occupancy is 1 at $(0,0)$ and is 0 everywhere else. The `cell_value` is 0 for all indices. **You should only call this function once (at the beginning of your program)**.

`int advance_frame(int axis, int direction, GameBoard *gameBoard)`

> Updates the `occupancy`, `cell_value`, and `snek` of `gameBoard` after advancing by 1 frame given the `axis` (one of {`AXIS_X`, `AXIS_Y`}), direction (one of {`UP`, `DOWN`, `LEFT`, `RIGHT`}) and the current `gameBoard`. Returns 0 if the `gameBoard` is now in a failure state, and 1 otherwise. Use this function to control your `Snek`.

`void show_board(GameBoard *gameBoard)`

> Displays the board's current state. By continuously running `advance_frame()` followed by `show_board()` in a loop controlled by the return value of `advance_frame()`, you can use the nifty visual interface seen in figure 1.

`void end_game(GameBoard *gameBoard)`

> Shows the game over screen and **cleans all used memory**. You **should always run `end_game()` before your main function terminates** otherwise there will be memory leaks.

VOLATILE CONSTANTS

The following are constants defined in snek_api.h which are central to the operation of the game. It is recommended that you do not modify these.

```
#define LIFE_SCORE 1 //score awarded for staying alive one frame

#define AXIS_X -1
#define AXIS_Y 1

#define UP -1
#define DOWN 1
#define LEFT -1
#define RIGHT 1

#define AXIS_INIT AXIS_Y
#define DIR_INIT DOWN

#define x 0          //for indexing the SnekBlock
#define y 1

#define MOOGLE_POINT 20
#define HARRY_MULTIPLIER 3
```