STATE: Initial config

- Set default config values
- Ask for username

OUT: Username

STATE: (Re)Start

- Direct user to the menu they want (game, TOP scores, settings)

Quit

STATE: TOP Scores

IN: Username

- Show TOP scores
- Show score of the user

STATE: Settings

IN: Config

- Show list of configurable things and their state
- Ask for changes or start

OUT: Config

STATE: Playing

IN: Config

- Initialize game
- Get key inputs, update & print periodically, pause and resume

OUT: Score

STATE: Game Over

IN: Username, score

- Free heap space
- Save results in the history log AND TOP score

Initial config
OUT: game_config, username

Create and initialize w/default values the game_config structure

Ask for username (stdin) & save it

(Re)Start

Print menu & ask for next menu (TOP scores, settings, Play)

TOP Scores

Open TOP scores files and load the data

Print the data to the screen

Wait for user signal (stdin) to exit the menu

STATE: Settings
IN: Config
OUT: Config

Print settings list

Ask for user input (stdin)

Update config

User input?

Quit

...

Playing

...

(Re)Start

Playing

IN: config, username
LOCALS: snake (heap), food, score
OUT: score

Initialize snake, food & score

Print header
Print board

Wait 1 gametick while checking key input

User input?

Key input?

Arrow or none

Update snake

Ate food?
Collision?

Ate food

Snake growth
New food
Update score

None

Collision

Print game over animation & free heap

Print game over screen & wait for user signal (stdin)

Pause key

- Print pause menu
- Ask for quit or resume (stdin)

Open history log & append this game

New score > some TOP score?

Open TOP scores & read the data

Save the new TOP scores to the file

Go to start menu

Graphics

- void printGameInit(const part_t* phead, uint width, uint height, food_t food);
- void printBoard(const part* phead, food_t food, uint width, uint height);
  Prints the board of the game
- void printHeader(const char user[], uint score, uint lives);
- void printGameOver(uint score);
- void eraseScreen();

Game rules

- Typedef enum { UP = N, DOWN = S, …, PAUSE } key_t;
- key_t getInputInGametick (uint gametick_length);
  Checks for a key input during a whole gametick

typedef struct { uint x; uint y } food_t;

- food_t newFood(uint width, uint height);
- Macro OCURREDCOLLISION(phead)

- gamestate_t runOneGameTick(snake_t* snake, food_t* food, uint* score, const info_t* gameinfo);

Menus

- gamestate_t pause();
  Prints the pause menu and waits for user input to resume or exit, and returns it
- void initGame(settings_t* settings, char* user_buffer);
  Initializes the default settings and asks the player for the name
- gamestate_t printStart();
  Prints the start module and asks for the menu (play, TOP scores or settings)
- void printTopScore(const char* users_buffer[], const uint scores_buffer[]);

Body

- int initSnake(snake_t* snake, int init_x, int init_y, direction_t init_orient, uint init_length);
  Initializes the snake with some initial position, orientation and length.
  Returns 0 on defult or error code if malloc fails.
- int isInsideSnake(int x, int y, const part_t* phead);
  Boolean function: returns 1 if there's a part of the snake in the position (x, y), 0 if not
- void update(snake_t* snake, direction_t dir);
  Moves the snake one step into the direction dir, not increasing it's size
- int newNode(snake_t* snake);
  Creates a new part behind the tail and saves it in the snake as the new tail.
  Returns 0 on defult or error code if malloc fails
- void freeAll(part_t* phead);
  Frees the memory used for the snake on the heap.

progdata.h

```
typedef struct {
        uint board_width;
        uint board_height;
        uint init_snake_length;
        clock_t tick;
        char username[MAX_USER_LENGTH];
        …
} settings_t;

typedef enum {
        PLAYING, PAUSE, GAMEOVER,  START,
TOPSCORES, SETTINGS
} gamestate_t;

// + constants and other things
```

History

```
int saveThisMatch(char user[], uint score);

int appendMatchToTopScore(char user[], uint score);

int getTopScore(char* users_buffer, uint* scores_buffer);
```