```
               _  __
 ___ _ __   ___| |/ /
/ __| '_ \ / _ \ ' / 
\__ \ | | |  __/ . \ 
|___/_| |_|\___|_|\_\
```

The Adventures of Ormis

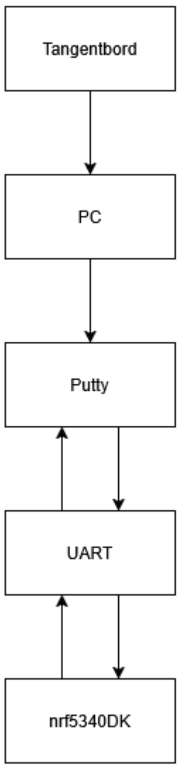
Hur man spelar

Spelet behöver en baudrate på 1 000 000 för att hinna med att skriva ut all grafik.

I putty:
Local echo: forced off
Local echo: forced off

Använd WASD för att styra din figur. Kör över äpplen för att äta dem och växa. Det är jätteonödigt att krocka. Om du inte följer rekommendationen att undvika kollisioner kan du trycka på ‘r’ för att starta om.

Systemöverblick



Datorn tar emot tangent inputs från tangentbordet. Putty används för att läsa inputsen och skickar dem vidare med UART till kortet. Kortet kör spelet och skickar tillbaka output via UART. Outputten visas i Putty terminalen.

Implementation (pseudokod)

```
void send_int(int number)
{
        outputs an int in putty
}

void sendText(char *str)
{
        outputs a string to putty
}

void send_breakline()
{
        makes a new line in putty
}

void createGameBoard(struct Square *gameBoardArray)
{
i is 0

FOR (y is 0; y smaller than BOARD_HEIGHT; y becomes larger by 1)
{
        FOR (x is 0; x smaller than BOARD_WIDTH; x becomes larger by 1)
        {
                gameBoardArray[i].xCoord is x
                gameBoardArray[i].yCoord is y

                IF (y is 0 OR x is 0 OR y  BOARD_HEIGHT - 1 OR x is BOARD_WIIDTH -1)
                        gameBoardArray[i]occupier is 1
                ELSE
                        gameBoardArray[i]occupier is 0
                i becomes larger by 1
        }
}
}
int findBoardIndex(struct Square *gameBoardArray, int x, int y)
{
FOR (i is 0; i smaller than BOARD_HEIGHT*BOARD_WIDTH; i becomes larger by 1)
        {
                IF(gameBoardArray[i].xCoord is x AND gameBordArray[i].yCoord is y)
                        return i
        }
}

void changeOccupier(struct Square *gameBoardArray, int x, int y, int type)
{
i is findBoardIndex(gameBoardArray, x, y)
SWITCH (type)
{
        CASE 0:
                gameBoardArray[i].occupier is backgrund
        CASE 1:
                gameBoardArray[i].occupier is border
        CASE 2:
                gameBoardArray[i].occupier is snake
        CASE 3:
                gameBoardArray[i].occupier is apple
}
}

void appleSpawner(struct Square *gameBoardArray)
{
WHILE(1)
{
        boardArraySize = BOARD_HEIGHT * BOARD_WIDTH
        i = randomized number smaller than boardArraySIZE

        IF(gameBoardArray[i].occupier is 0)
        {
                changeOccupier(gameBoardArray, gameBoardArray[i].xCoord, gameBoardArray[i].yCoord, 3)
        }
}
}

static struct SnakeHead checkCollision(struct Square *gameBoardArray, struct SnakeHead head,
struct SnakeBody *body)
{
SWITCH (head.direction)
{
        CASE 0:
                IF (next position to right is border OR snake)
                        snake is dead :(
                ELSE IF(next position right is an apple)
                {
                        head = eatApple(head, body)
                        eatedIt = 1
                }
        CASE 1:
                IF (next position to down is border OR snake)
                        snake is dead
                ELSE IF(next position down is an apple)
                {
                        head = eatApple(head, body)
                        eatedIt = 1
                }
        CASE 2:
                IF (next position to left is border OR snake)
                        snake is dead
                ELSE IF(Next position to left is apple)
                {
                        head = eatApple(head, body)
                        eatedIt = 1
                }
        CASE 3:
                IF (next position to up is border OR snake)
                        snake is dead
                ELSE IF(next position to up is apple)
                {
                        head = eatApple(head, body)
                        eatedIt = 1
                }
}
return head
}

static struct SnakeHead moveBody(struct Square *gameBoardArray, struct SnakeHead head, struct
SnakeBody *body)
{
        FOR(i = length of snake; i >= 0; i--)
        {
                IF(last body piece)
                        Change occupier of current position
                IF(first body piece behind head)
                        update body position to heads position
                ELSE IF(last body piece and has just eaten apple)
                {
                        don't move
                        eatedIt = 0
                }
                ELSE
                        update position to body piece ahead
        }
}

static struct SnakeHead moveHead(struct Square *gameBoardArray, struct SnakeHead head)
{
        SWITCH(head.direction)
        CASE 0 //right
                increase head x coord with one
                changeOccuiper() at the new position.
        CASE 1 //down
                increase head y coord with one
                changeOccuiper() at the new position.
        CASE 2 //left
                decrease head x coord with one
                changeOccuiper() at the new position.
        CASE 3 //up
                decrease head y coord with one
                changeOccuiper() at the new position.
}

struct SnakeHead createSnakeHead(int startX, int startY)
{
        create a new empty SnakeHead
        fill start position given to the function
        direction = 0
        length = 0
        alive = 1
}

void createSnakeBody(struct SnakeBody *bodyArr, int x, int y, int place)
{
        create a new empty SnakeBody
        fill all variables to the ones inputted with the function
}

struct SnakeHead changeDirection(struct SnakeHead head)
{
        SWITCH(uarte_interupt_buffer)
        CASE ‘w’
        head.direction = 3
        CASE ‘s’
        head.direction = 1
        CASE ‘a’
        head.direction = 2
        CASE ‘d’
        head.direction = 0
}

struct SnakeHead eatApple(struct SnakeHead head, struct SnakeBody *body)
{
        createSnakeBody() create a body att the end of the snake
        increase head.lenth
        decrease currentApples.
}

struct SnakeHead moveSnake(struct Square *gameBoardArray, struct SnakeHead head, struct
SnakeBody *body)
{
        checkCollision() check for collision

        IF(snake is alive)
        {
                IF(snake has a body)
                        moveBody()
                moveHead()
        }

}

void printLogo()
{
        uses sendText() and sendBreakline() to write out the sneK logo
}

void printGameOver()
{
```

```
          uses sendText() and sendBreakline() to write out the game over screen
}

void printGameBoard(struct Square *gameBoardArray)
{
          define different color pixels by setting background color of two blankspaces

          FOR(i = 0; i < number of squares in the gameboard array; i++)
          {
                    SWITCH(gameboard[i].occupier)
                    CASE 0
                    sendText(backgorund)
                    CASE 1
                    sendText(border)
                    CASE 2
                    sendText(snake)
                    CASE 3
                    sendText(apple)
          }
}

void start_game_grupp12(void);
{
NRFX startup
rtc config

uarte config

WHILE (TRUE)
{
          IF(playAgain == 1)
          {
                    int startPosX = starting point
                    int startPosY = starting point

                    struct Square gameBoardArray[BOARD_WIDTH * BOARD_HEIGHT];
                    createGameBoard(gameBoardArray);

                    struct SnakeHead snakeHead;
                    snakeHead = createSnakeHead(startPosX, startPosY);
                    changeOccupier(gameBoardArray, snakeHead.xCoord, snakeHead.yCoord, 2);

                    struct SnakeBody snakeBody[Number of squares on the board (max possible snake
                    length)];

                    create the first body parts of the snake
                    createSnakeBody()
                    createSnakeBody()
                    snakeHead.length  = 2;

                    Game logic
                    WHILE(snake is alive)
```

```
        {
            nrfx_uarte_rx() For interrupts

            clear the putty terminal

            printLogo()

            write out score and previous best score

            changeDirection() update the snakes direction if

            moveSnake() move the snake in the current direction

            IF(currentApples < maxApples)
                appleSpawner() Spaw a new random apple.

            printGameBoard() write out the game board to the putty terminal
        }

        playAgain = 0
        printGameOver()
        print out score and highscore
        prompt player to restart
    }
    handle keyboard inputs to restart
    if(key press == r)
    {
        play again = 1
        currentApples = 0
    }
}
}
```