

Daniel Henrique Nakazawa R.A. 001201902712

João Victor Domingues Rizzardo R.A. 001201908443

Universidade São Francisco

Computação Gráfica e Processamento de Imagens

Trabalho N2

Jogos Digitais

Bragança Paulista

2022

Introdução

Um jogo digital, se refere a jogos eletrônicos desenhados para serem jogados em um computador, console ou outro dispositivo tecnológico, como celulares. Segundo Crawford, existem quatro elementos fundamentais de todos os jogos: representação, interação, conflito e segurança.

Representação: o jogo fornece uma representação simplificada e subjetiva da realidade, tendo um conjunto de regras explícitas.

Interação: Nela, o expectador é capaz de provocar alterações e verificar suas consequências, sendo assim capaz de modificar a realidade apresentada.

Conflito: O conflito surge naturalmente a partir da interação do jogador e esse elemento está presente em todos os jogos.

Segurança: O jogo permite que o jogador se submeta à experiência psicológica do conflito e do perigo sem os danos físicos, possibilitando assim desassociar as consequências das ações.

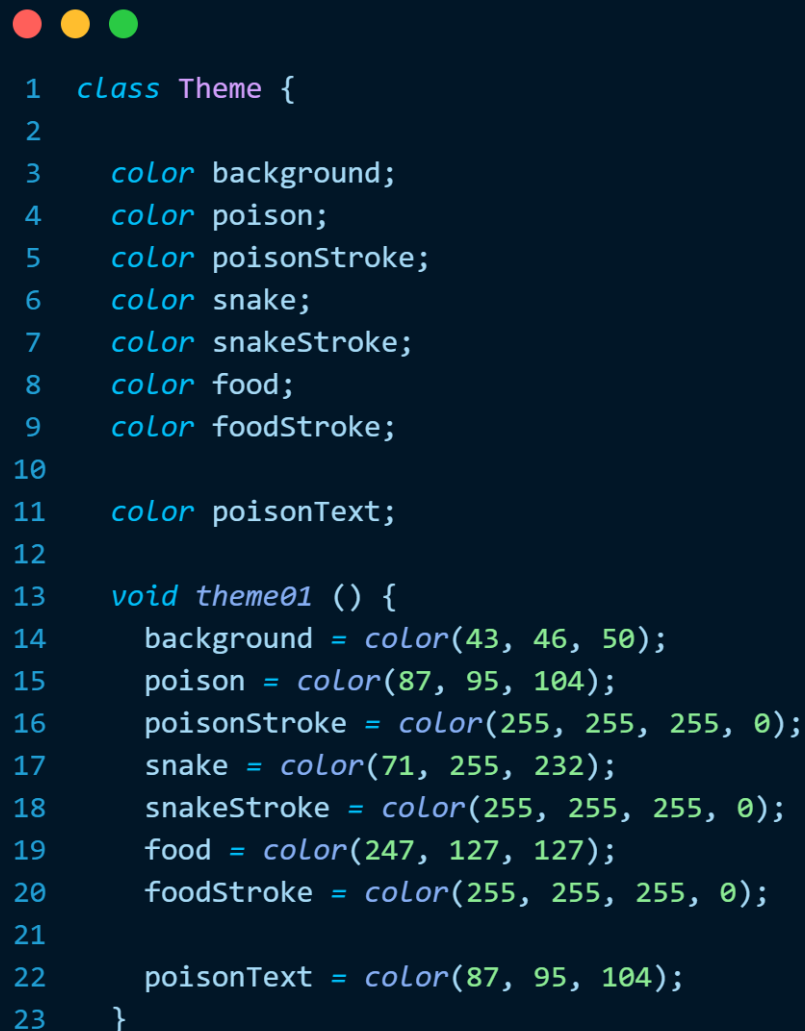
No nosso trabalho tentamos recriar um dos jogos mais famosos da história, estamos falando do Snake Game. O jogo consiste em controlar uma cobra, a qual deve percorrer o cenário e ir se alimentando, cada vez que a cobra é alimentada é obtido um ponto e o tamanho da cobra é aumentado, se a cobra se colidir com uma das bordas do cenário ou em seu próprio corpo, o jogo é finalizado e você perde. Em nossa versão a cada 5 pontos o jogador sobe um level, e a um level uma unidade de veneno é adicionada no mapa, se a cobra colidir com um veneno o jogo também é perdido.

Metodologia

Nosso projeto foi realizado baseado em classes, a seguir mostrarei um pouco sobre cada uma delas.

Classe Theme:

Responsável por aplicar temas no jogo, cada função de seu escopo é referente a um tema diferente.



```
1  class Theme {
2
3      color background;
4      color poison;
5      color poisonStroke;
6      color snake;
7      color snakeStroke;
8      color food;
9      color foodStroke;
10
11     color poisonText;
12
13     void theme01 () {
14         background = color(43, 46, 50);
15         poison = color(87, 95, 104);
16         poisonStroke = color(255, 255, 255, 0);
17         snake = color(71, 255, 232);
18         snakeStroke = color(255, 255, 255, 0);
19         food = color(247, 127, 127);
20         foodStroke = color(255, 255, 255, 0);
21
22         poisonText = color(87, 95, 104);
23     }
```

Classe Food:

Responsável por abstrair a food (comida) da cobra e tratar de algumas funções relacionadas ao veneno.

```
1 class Food {
2     PVector food = new PVector((fLooR(random(b.boardWidth))), (fLooR(random(b.boardHeight))));
3     ArrayList<PVector> poison = new ArrayList<PVector>();
4
5     int currentLevel = 0;
6
7     void render() {
8
9         stroke(theme.foodStroke);
10        fill(theme.food);
11        rect(food.x*scl, food.y*scl, scl, scl);
12
13        // CADA VEZ QUE SUBIR UM LEVEL ADICIONA UM VENENO
14        if (currentLevel == 1) {
15            poison.add(new PVector((fLooR(random(b.boardWidth))), (fLooR(random(b.boardHeight))));
16            for (int i = 0; i < poison.size(); i++) {
17                rect(poison.get(i).x*scl, poison.get(i).y*scl, scl, scl);
18            }
19
20            currentLevel = 0;
21        }
22
23        stroke(theme.poisonStroke);
24        fill(theme.poison);
25        for (PVector p : poison) {
26            rect(p.x*scl, p.y*scl, scl, scl);
27        }
28
29
30    }
```

Classe GameBoard:

Responsável por abstrair a área que a cobra percorre, ou seja, a interface jogável.

```
1  class Board {
2      int boardWidth;
3      int boardHeight;
4      int currentScore = 0;
5
6      Board(int w, int h) {
7          boardWidth = w;
8          boardHeight = h;
9      }
10
11     void render() {
12         noStroke();
13         fill(theme.background);
14         rect(0, 0, b.boardWidth*scl, b.boardHeight*scl);
15     }
16
17     void level() {
18         // A CADA 5 PONTOS SOBE UM LEVEL
19         if (currentScore == 5) {
20
21             level++;
22
23             // VARIÁVEL UTILIZADA PARA GERAR O VENENO
24             f.currentLevel++;
25             currentScore = 0;
26         }
27     }
28
29 }
30
```

Classe Menu:

Responsável por abstrair as screens (telas), ou seja, as interfaces do jogo.

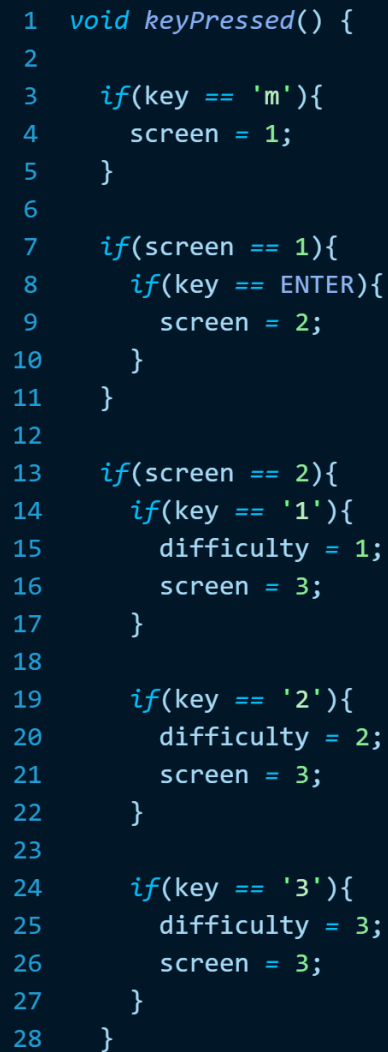
```
1  class Menu {
2
3      void screen1(){
4          PImage img = loadImage("data/screen1.png");
5
6          image(img, 0,0);
7      }
8
9      void screen2(){
10         PImage img = loadImage("data/screen2.jpg");
11
12         image(img, 0,0);
13     }
14
15     int changeDifficulty(int difficulty){
16         int rate = 0;
17
18         switch(difficulty){
19             case 1:
20                 rate = 10;
21                 break;
22
23             case 2:
24                 rate = 15;
25                 break;
26
27             case 3:
28                 rate = 25;
29                 break;
30         }
31
32
33     }
```

Classe Snake:

Responsável por abstrair a cobra e suas funções.


```
1 class Snake {
2   PVector snakeLoc = new PVector((floor(random(b.boardWidth))), (floor(random(b.boardHeight))));
3
4   float xDir = 1;
5   float yDir = 0;
6   int total = 0;
7
8   ArrayList<PVector> body = new ArrayList<PVector>();
9
10  void render() {
11    fill(theme.snake);
12    stroke(theme.snakeStroke);
13    rect(snakeLoc.x*scl, snakeLoc.y*scl, scl, scl);
14
15    for (int i = 0; i < body.size(); i++) {
16      rect(body.get(i).x*scl, body.get(i).y*scl, scl, scl);
17    }
18  }
19 }
20
21 boolean eat() {
22   float d = dist(snakeLoc.x, snakeLoc.y, f.food.x, f.food.y);
23
24   if (d < 1) {
25     pointEffect.rewind();
26     pointEffect.play();
27
28     total++;
29     return true;
30   } else {
31     return false;
32   }
33 }
```

Como dito anteriormente, o jogo foi baseado em screens, a principal motivação para isso, foi o fato de o jogo ser totalmente controlado pelo teclado, e devido a isso algumas teclas estavam conflitando com as funcionalidades de outra tela, logo algumas teclas exercem sua funcionalidade dependendo de qual screen o jogo esteja. A seguir um pequeno exemplo do que foi descrito.



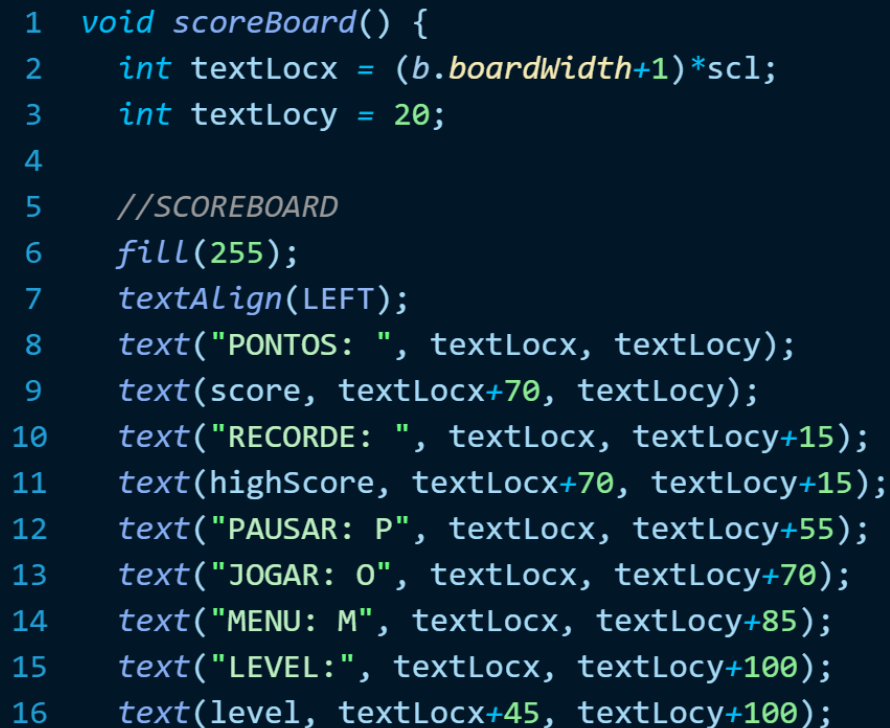
```
1 void keyPressed() {  
2  
3     if(key == 'm'){  
4         screen = 1;  
5     }  
6  
7     if(screen == 1){  
8         if(key == ENTER){  
9             screen = 2;  
10        }  
11    }  
12  
13    if(screen == 2){  
14        if(key == '1'){  
15            difficulty = 1;  
16            screen = 3;  
17        }  
18  
19        if(key == '2'){  
20            difficulty = 2;  
21            screen = 3;  
22        }  
23  
24        if(key == '3'){  
25            difficulty = 3;  
26            screen = 3;  
27        }  
28    }
```


Os níveis de dificuldade do jogo foram definidos em: Fácil, Normal e Difícil. O método para aumentar ou diminuir a dificuldade foi aumentar a velocidade com que a cobra percorre a tela. Para isso manipulamos o framerate do jogo. Inicialmente tentamos aumentar a quantidade de pixels que a cobra se movia por frame, mas isso estava prejudicando a jogabilidade, devido a isso, podemos perceber que alterando o framerate teríamos um resultado mais satisfatório. Essa funcionalidade é realizada pelo método `changeDifficulty` da classe `Menu`.



```
1  int changeDifficulty(int difficulty){
2      int rate = 0;
3
4      switch(difficulty){
5          case 1:
6              rate = 10;
7              break;
8
9          case 2:
10             rate = 15;
11             break;
12
13          case 3:
14              rate = 25;
15              break;
16     }
17
18     return rate;
19 }
```

Na interface do jogo, possuímos um menu lateral que exibe as informações do jogo, como pontuação, level, recorde e algumas informações de utilidade, como teclas de atalho e uma legenda para o que é cada objeto na tela. Esse menu lateral é gerado pela função `scoreBoard`.



```
1 void scoreBoard() {
2     int textLocx = (b.boardWidth+1)*scl;
3     int textLocy = 20;
4
5     //SCOREBOARD
6     fill(255);
7     textAlign(LEFT);
8     text("PONTOS: ", textLocx, textLocy);
9     text(score, textLocx+70, textLocy);
10    text("RECORDE: ", textLocx, textLocy+15);
11    text(highScore, textLocx+70, textLocy+15);
12    text("PAUSAR: P", textLocx, textLocy+55);
13    text("JOGAR: O", textLocx, textLocy+70);
14    text("MENU: M", textLocx, textLocy+85);
15    text("LEVEL:", textLocx, textLocy+100);
16    text(level, textLocx+45, textLocy+100);
```

Como dito anteriormente, a jogo possui níveis, o level tem o seguinte funcionamento: A cada 5 pontos ganhos, isto é, cada vez que a cobra se alimentar 5 vezes um level é incrementado. Essa funcionalidade é realizada pelo método level da classe GameBoard.

```
1 void level() {
2     // A CADA 5 PONTOS SOBE UM LEVEL
3     if (currentScore == 5) {
4
5         level++;
6
7         // VARIÁVEL UTILIZADA PARA GERAR O VENENO
8         f.currentLevel++;
9         currentScore = 0;
10    }
11 }
```

Iremos agora falar um pouco sobre nossa classe Snake, que busca abstrair a cobra. A posição da cobra é armazenada inicialmente em uma variável do tipo PVector, que armazena as posições em x e y da cobra. Essa posição é inicialmente dada de forma randômica dentro da área jogável. O corpo da cobra é armazenado em uma variável do tipo ArrayList, dentro dessa lista é armazenada diversas variáveis do tipo PVector que será correspondente as posições de cada parte do corpo da cobra.

```
1 class Snake {
2     // POSIÇÃO INICIAL RANDOMICA DA COBRA
3     PVector snakeLoc = new PVector((float)random(b.boardWidth)), (float)random(b.boardHeight));
4
5     // DIREÇÃO EM X QUE A COBRA ESTÁ PERCORRENDO
6     float xDir = 1;
7
8     // DIREÇÃO EM Y QUE A COBRA ESTÁ PERCORRENDO
9     float yDir = 0;
10
11     // TAMANHO TOTAL DA COBRA
12     int total = 0;
13
14     // VARIÁVEL CONTENDO O CORPO DA COBRA
15     ArrayList<PVector> body = new ArrayList<PVector>();
```

O método render é responsável por percorrer o corpo da cobra e renderizar na tela.

```
1 void render() {
2   fill(theme.snake);
3   stroke(theme.snakeStroke);
4   rect(snakeLoc.x*scl, snakeLoc.y*scl, scl, scl);
5
6   for (int i = 0; i < body.size(); i++) {
7     rect(body.get(i).x*scl, body.get(i).y*scl, scl, scl);
8   }
9
10 }
```

O método eat é responsável por verificar se a cobra comeu a comida, ele é baseado na função dist, nativa do processing, que calcula a distância entre dois pontos, se a distância entre a cobra e a comida for menor que um, ele retorna verdadeiro e incrementa em um a variável total, utilizada para gerar o corpo da cobra, nesse trecho do código foi utilizado um efeito sonoro para quando um ponto for adquirido, esse efeito foi implementado utilizando a biblioteca de áudios Minim.

```
1 boolean eat() {
2   float d = dist(snakeLoc.x, snakeLoc.y, f.food.x, f.food.y);
3
4   if (d < 1) {
5     pointEffect.rewind();
6     pointEffect.play();
7
8     total++;
9     return true;
10  } else {
11    return false;
12  }
13 }
```

Agora iremos falar a respeito do método `death`, responsável por executar a morte da cobra. Esse método é trabalhado em duas condições, a primeira, é percorrido o corpo da cobra, e em cada segmento é verificado a distância entre o mesmo e o primeiro segmento, que corresponde à frente da cobra, se essa distância for menor que um, teremos a execução da função `gameover`, que resulta em fim de jogo.

A segunda condição é executada apenas quando o level for maior que um, significando a existência de venenos pelo mapa, é verificado a distância da cobra para cada veneno existente, e se em uma dessas ocorrências a distância for menos que um, também resultará em `gameover`.

```
1 void death() {
2
3     // VERIFICA SE A COBRA ESTÁ COLIDINDO COM SEU PRÓPRIO CORPO OS AS BORDAS DO MAPA
4     for (int i = 0; i < body.size(); i++) {
5         PVector pos = body.get(i);
6         float d = dist(snakeLoc.x, snakeLoc.y, pos.x, pos.y);
7         if (d < 1) {
8             gameOver();
9         }
10    }
11
12    // VERIFICA SE A COBRA COLIDIU COM UM VENENO
13    if (level > 0) {
14        float p;
15        for (int i = 0; i < f.poison.size(); i++) {
16            p = dist(snakeLoc.x, snakeLoc.y, f.poison.get(i).x, f.poison.get(i).y);
17
18            if (p < 1) {
19                gameOver();
20            }
21        }
22    }
23 }
```

O método `gameover` executa um efeito sonoro com a biblioteca `Minim` e renderiza a screen de `gameover`.

```
1 void gameOver() {
2     gameOverEffect.rewind();
3     gameOverEffect.play();
4
5     fill(255, 0, 0);
6     textSize(30);
7     textAlign(CENTER);
8     text("GAME OVER", b.boardWidth/2*scl, b.boardHeight/2*scl);
9     textAlign(CENTER, TOP);
10    text("PRESSIONE 'R' PARA RECOMEÇAR", b.boardWidth/2*scl, b.boardHeight/2*scl);
11    body.clear();
12    noLoop();
13
14    if (score > highScore) {
15        highScore = score;
16    }
17    level = 0;
18 }
```

O método `update` é responsável por adicionar um novo elemento do tipo `PVector` ao `ArrayList` de corpo da cobra. Também é responsável por definir a direção em que a cobra está percorrendo de acordo com a tecla pressionada.

```
1 void update() {
2     if (total > 0) {
3         if (total == body.size()) {
4             body.remove(0);
5         }
6         body.add(new PVector(snakeLoc.x, snakeLoc.y));
7     }
8
9     snakeLoc.x += (1 * xDir);
10    snakeLoc.y += (1 * yDir);
11
12    snakeLoc.x = constrain(snakeLoc.x, 0, b.boardWidth-1);
13    snakeLoc.y = constrain(snakeLoc.y, 0, b.boardHeight-1);
14
15 }
```

Agora iremos falar a respeito da classe Food, que é responsável por tratar das funções referentes a comida e aos venenos. A comida é instanciada em uma variável do tipo PVector que irá armazenar sua posição em x e y de forma randômica. Já os venenos, como podem existir diversos espalhados pelo mapa, será armazenado em um ArrayList com seus elemento do tipo PVector. A função render é responsável por renderizar tanto as comidas quando os venenos.

```
1 void render() {
2
3     stroke(theme.foodStroke);
4     fill(theme.food);
5     rect(food.x*scl, food.y*scl, scl, scl);
6
7     // CADA VEZ QUE SUBIR UM LEVEL ADICIONA UM VENENO
8     if (currentLevel == 1) {
9         poison.add(new PVector((floor(random(b.boardWidth))), (floor(random(b.boardHeight)))));
10        for (int i = 0; i < poison.size(); i++) {
11            rect(poison.get(i).x*scl, poison.get(i).y*scl, scl, scl);
12        }
13    }
14    currentLevel = 0;
15 }
16
17 stroke(theme.poisonStroke);
18 fill(theme.poison);
19 for (PVector p : poison) {
20     rect(p.x*scl, p.y*scl, scl, scl);
21 }
22
23 }
```

O método `foodLocation` tem a seguinte funcionalidade, ele realiza uma comparação e ao mesmo tempo executa o método `eat` da classe `snake`, se esse método retornar `true`, ou seja, se for verdadeiro que a cobra comeu a comida, será incrementado em um na pontuação e a comida receberá uma nova posição randômica.



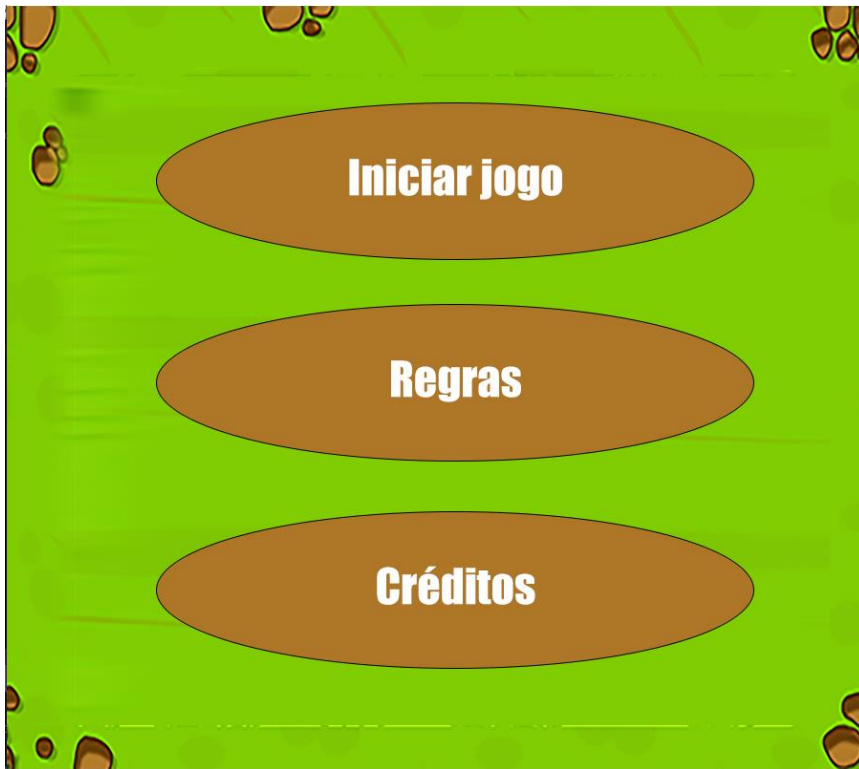
```
1 void foodLocation() {  
2  
3     if (snake.eat() == true) {  
4         score ++;  
5         b.currentScore++;  
6  
7         food.x = (floor(random(b.boardHeight)));  
8         food.y = (floor(random(b.boardWidth)));  
9     }  
10  
11     food.x = constrain(food.x, 0, b.boardHeight-1);  
12     food.y = constrain(food.y, 0, b.boardWidth-1);  
13  
14 }
```


Resultados

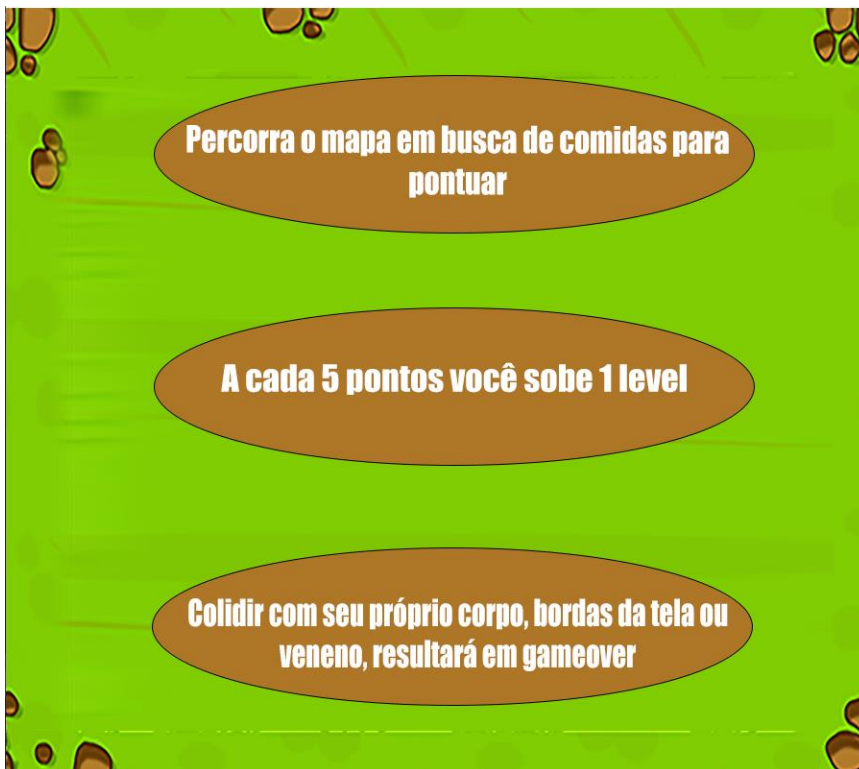
Tela inicial do jogo.



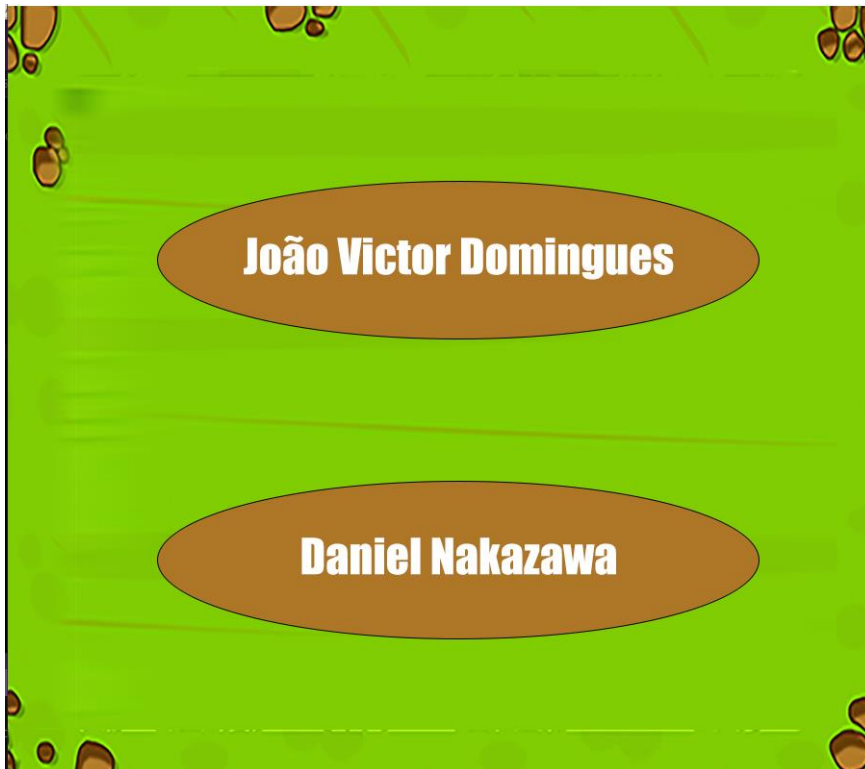
Após pressionar a tecla enter o jogador será direcionado para o menu do jogo.



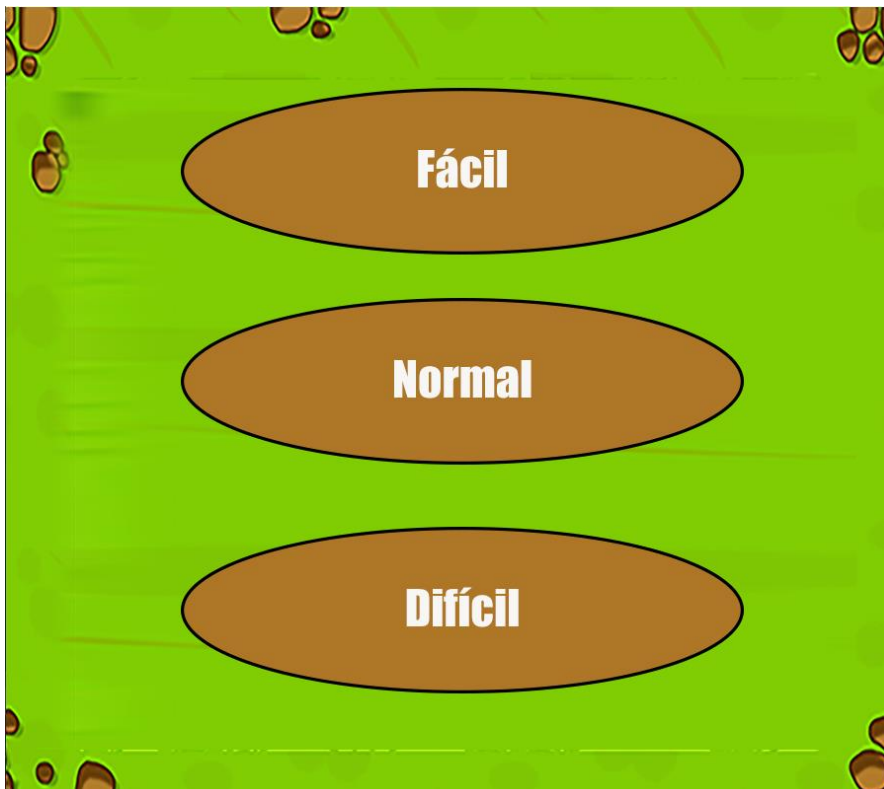
A navegação nos menus do jogo é realizada utilizando as teclas numéricas do teclado "1", "2", "3", nesse exemplo. Pressionando a tecla "2" o jogador será direcionado para a tela referente as regras.



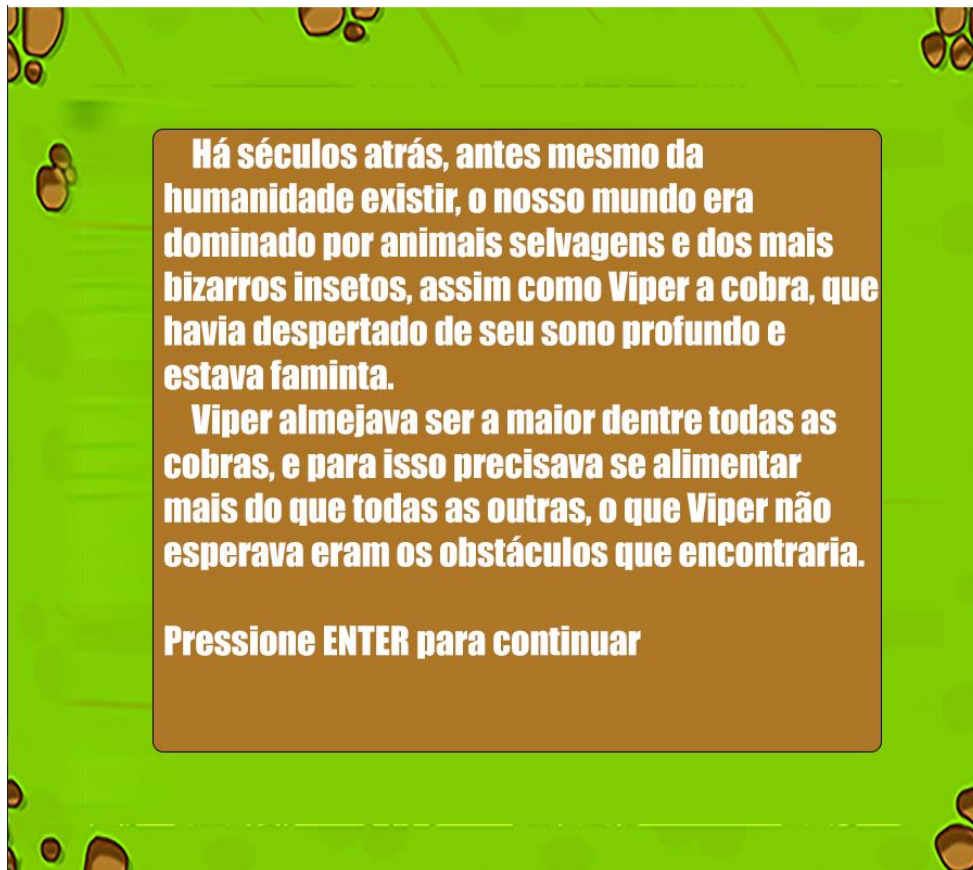
Para retornar ao menu principal será necessário pressionar a tecla “m”, essa tecla possui essa função em qualquer screen do jogo. Pressionando a tecla “3” o jogador é redirecionado para a tela de créditos.



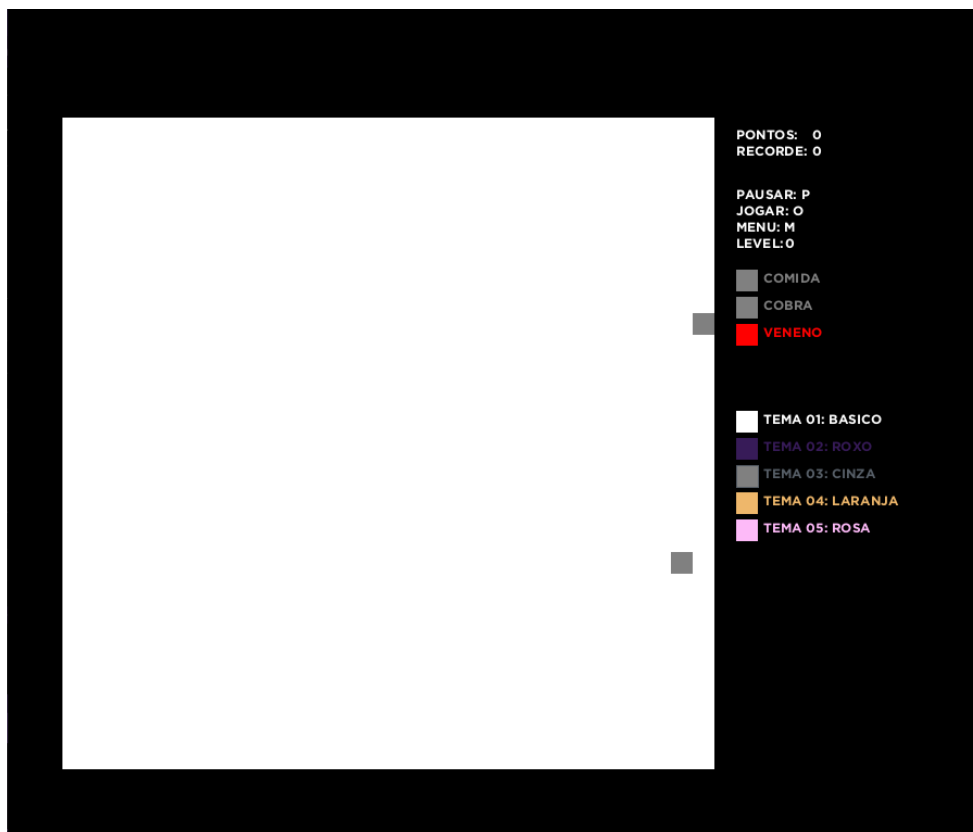
Por fim, se pressionado a tecla “1” o jogador será direcionado para tela das dificuldades, onde deverá escolher entre Fácil, Normal e Difícil, após selecionar a dificuldade será exibida a história do jogo.



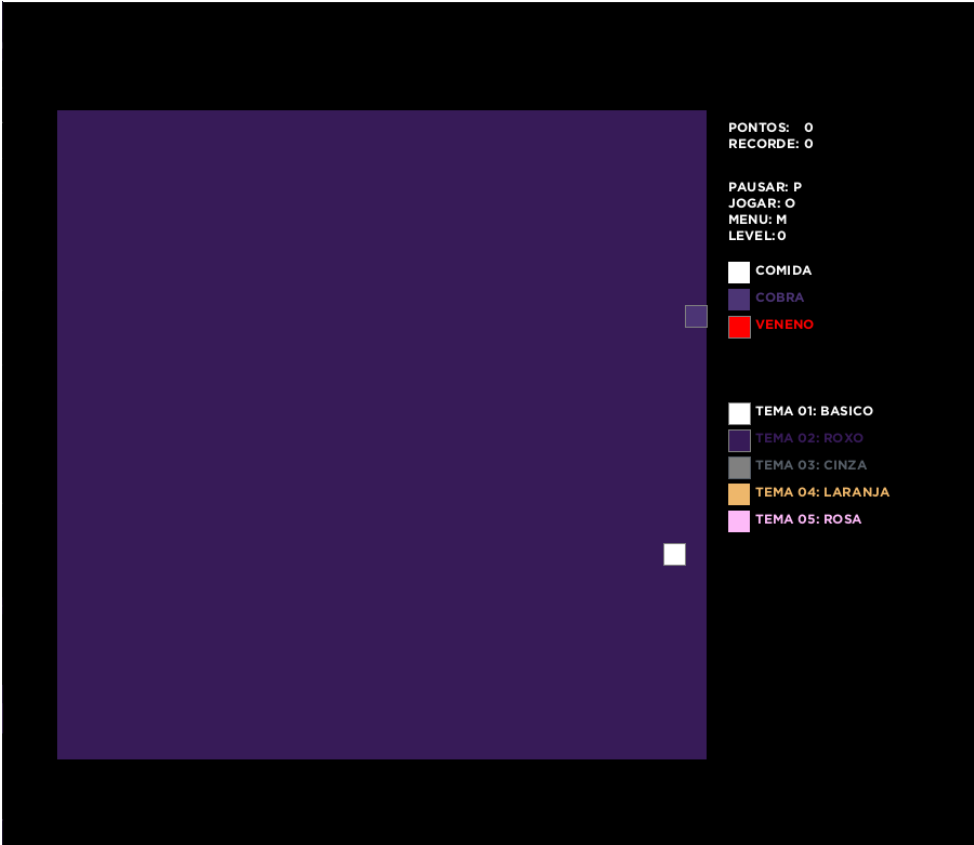
Será exibido a história do jogo, e pressionando a tecla ENTER o jogo será iniciado.



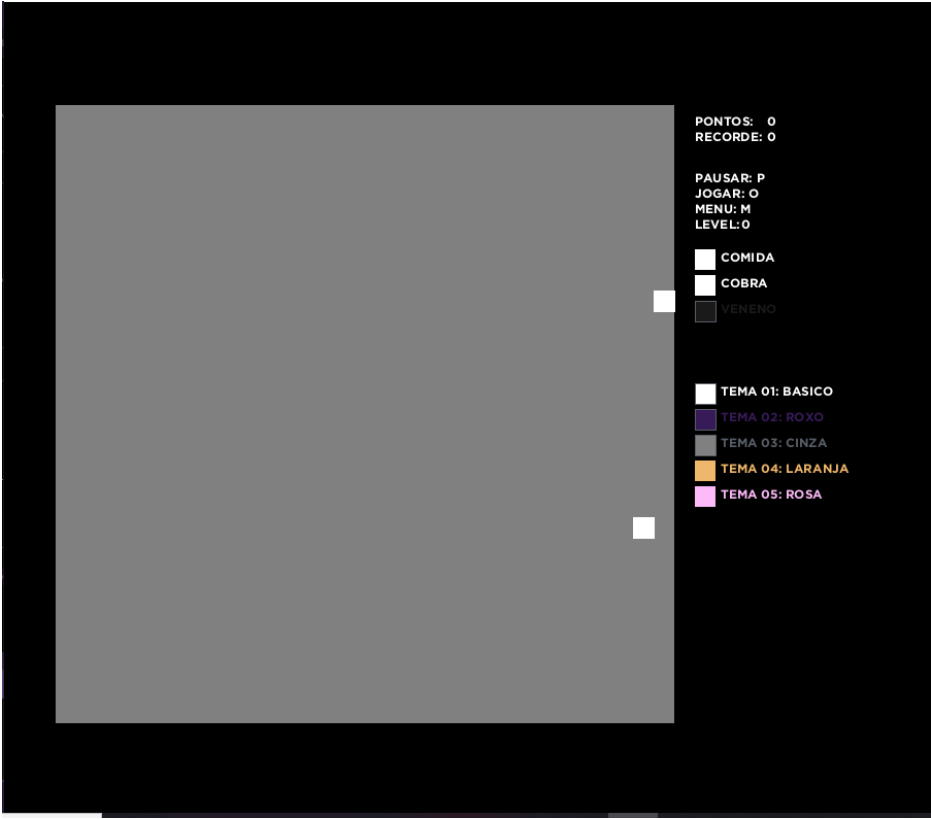
Jogo com o tema básico



Jogo com o tema roxo



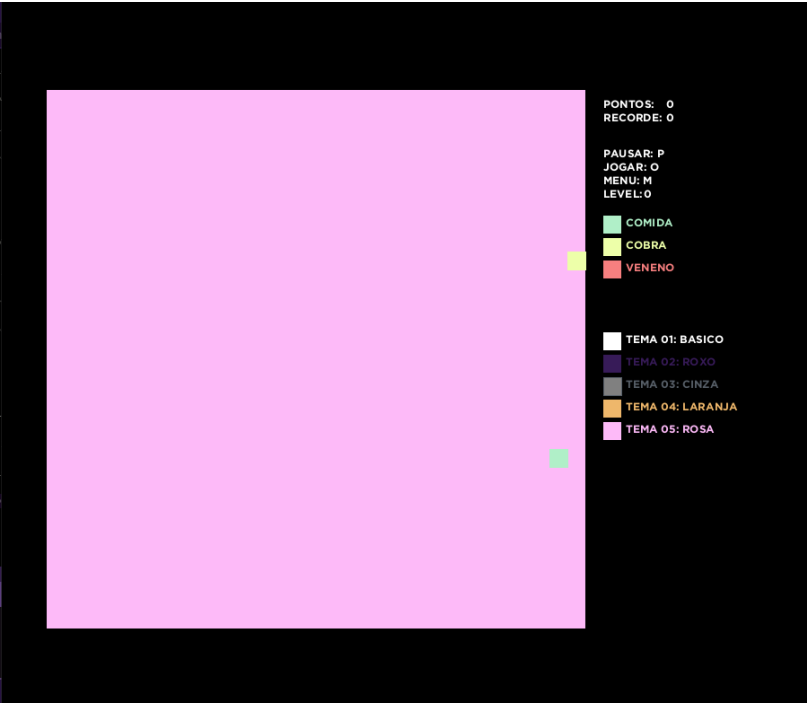
Jogo com o tema cinza



Jogo com o tema laranja



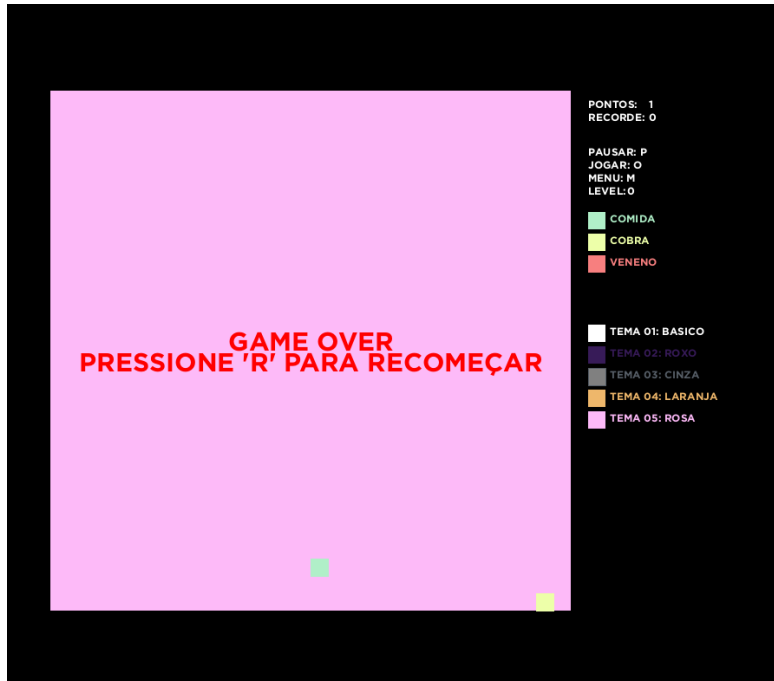
Jogo com o tema rosa



Os temas podem ser selecionados a partir do teclado numérico, de acordo com o número informado no menu lateral.

A tecla P é responsável por pausar o jogo, a tecla O para despausar, e a tecla M como já dito anteriormente, para retornar ao menu principal.

Tela de gameover



Link para o repositório no GitHub:

<https://github.com/joaov-rizzardo/snakeGame>

Conclusão

Podemos concluir que o projeto foi de extrema importância para aplicarmos grande parte dos conceitos vistos em aula, como transformações geométricas e o uso de bibliotecas de sons. Além disso, a execução desse projeto, nos permitiu conhecer um pouco sobre o vasto mundo dos games, onde a computação gráfica está amplamente presente. Portanto, ter essa experiência com uma atividade prática, nos permite ter uma visão mais ampla dos conceitos aprendidos em aula.

Referências

Conceituação de Jogos Digitais - Fabiano Lucchese e Bruno Ribeiro FEEC / Universidade Estadual de Campinas Cidade Universitária Zeferino Vaz, Campinas, SP, Brasil

<https://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g3.pdf>