Pedro Inácio Rodrigues Pontes

Prática 11
Laboratório de AEDs

Belo Horizonte, Brasil

2024

1 Introdução

O trabalho em questão trata-se de uma virtualização do jogo Torre de Hanoi. A sua implementação virtual foi feita utilizando-se da linguagem de programação Processing Java. As regras do jogo são:

- Apenas um disco pode ser movido por vez
- Somente o disco superior de uma pilha pode ser transferido para o topo de outra pilha ou para uma barra vazia
- Discos maiores não podem ser empilhados sobre discos menores

O objetivo dessa prática foi aplicar os conhecimentos sobre Stacks, adquiridos na aula nº 11 de AEDS. Percebe-se que a torre tem um funcionamento bastane similar a uma stack, pois apenas o elemento mais recente/superior da torre pode ser movido, enquanto mais antigos/inferiores só são movidos após todos os mais recentes/superiores que eles serem retirados.

2 Desenvolvimento

Para criar essa virtualização, foram utilizadas 4 classes:

- Timer
- Pillar
- Disc
- Main

2.1 Timer

Registra o tempo. Visualiazação do seu código principal:

```
public void update(){
  iterations ++;
  if (iterations % framerate == 0){
    seconds ++;
  }
  if (seconds == 60){
    seconds = 0;
    minutes ++;
```

```
    }
    formatedTime = String.format("%02d:%02d", minutes, seconds);
  }
```

2.2 Pillar

Cria os pilares sobre os quais os discos podem ser empilhados. O construtor da classe foi criado para ela ser inicalizada por meio de iteração e guardada num array de classe - tal coisa simplifica e enxuta os códigos que usam atributos e métodos dos 3 pilares. Como no jogo existem tres pilares i $<$ 3 na iteração. Visão do construtor:

```
public Pillar (int i) {
    this.Width = 10;
    this.Height = 490;
    this.xPos = 142.5 + i * (142.5 + Width);
    this.yPos = 100;
  }
```

O cálculo da posição x dos pilares é feito a partir do cálculo presente acima baseado nos valores de i. No fim, serão criados três pilares igualmente espaçados.

Há um método show() para exibir o pilar na interface gráfica

2.3 Disc

Cria os discos usados na torre. Segue o mesmo princípio de inicialização por meio de iteração sob algum comando *ex. for* e armazenamento em array de classe. Visão do construtor:

```
public Disc (int i, float size){
    this.Width = size;
    this.Height = Width/3;
    this.Width -= i*10;
    this.xPos = (width/2 - Width - 5)/2;
    this.yPos = height - (Height * (i + 1)) - 10;
    this.color1 = random(255);
    this.color2 = random(255);
    this.color3 = random(255);
  }
```

Aqui, são inicializadas randomicamente as cores e são feitos diversos cáculos para definir *Width, Height, xPos, yPos*, tendo como base os parâmetros i e size. size é utilizada para randomizar

as dimensões dos discos, mais especificamente do disco base, o qual influencia o valor de todos os outros. Se size fosse definida dentro do construtor, todos os discos teriam tamanhos randomizados, fazendo com que o ordenamento de maior embaixo e menor em cima seja perdido.

Também é feito um construtor para quando é necessário apenas inicializar a classe, sem inicializar seus valores, exceto Width:

```
public Disc(float size){
    this.Width = size;
}
```

É inicializado Width para poder manter um ordenamento desses discos com atributos não inicializados a partir de seu tamanho.

Há um método show() para exibir o objeto disc na interface gráfica

2.4 Main

Classe principal, possui 9 funções. As quais são descritas nas próximas sub subseções Globalmente, são inicializadas 7 variáveis, das quais 6 desempenham papel essencial na dinâmica do jogo:

- int move
- int referencedDisc = 0
- int totalDiscs = 5
- float size = random(110,135)
- boolean win = false
- boolean menu = true
- boolean initialize = true

É instanciada a classe Timer, o array da classe Pillar e o array de tower, a qual é uma stack que contém discos.

2.4.1 void setup()

Função Principal. Inicializa o tamanho da interface gráfica do Processing onde será executado o código, as imagens e fontes, e é feito um for que instancia os pilares e towers:

```
for (int i = 0; i < 3; i++) {
  tower[i] = new Stack<>();
  pillar[i] = new Pillar(i);
```

2.4.2 void draw()

Função principal. Chama o restante das funções. O código é autoexplicativo:

```
void draw() {
  if (menu){
    initialMenu();
  }
  else{
    if(win == false){
      game();
    }
    else{
      finalMenu();
    }
  }

  //Verify if the player wins
  if (tower[1].size() == totalDiscs || tower[2].size() == totalDiscs){
    win = true;
  }
}
```

2.4.3 void initialMenu()

Carrega o menu inicial

2.4.4 void finaMenu()

Carrega o menu final

2.4.5 void game()

Inicializa os discos de torre. Eles são inicializados fora da setup() por haver a necessidade de esperar o jogador escolher o número desejado de discos para a partida. Fazer essa inicialização uma única vez exige um código um pouco mais complexo. Segue:

```
if (initialize) {
  for (int i = 0; i < totalDiscs; i++) {
    tower[0].push(new Disc(i,size));
    }
  initialize = false;
}
```

O timer é ativado até o fim da partida. São exibidos os pilares e discos. Se um disco for clicado pelo mouse, tal seguirá ele. Isso é feito pelo seguinte código, que tem relação direta com a função moveDisc():

```
Disc d = tower[referencedDisc].peek();
d.show();

//To the disc follows the mouse after it is mousePressed
if (move == 1) {
    d.xPos = mouseX - d.Width/2;
    d.yPos = mouseY - d.Height/2;
}
```

2.4.6 void moveDisc()

É chamada dentro de mousePressed(). Códigos que possibilitam o disco seguir o mouse após clicado e ir para certa torre quando há um clique posterior na sua área. Segue o código:

```
if (initialize == false){
  Disc d = tower[referencedDisc].peek(); //Disc that is been referenced
  Disc[] disc = new Disc[3];

  //Create a instance of peek disc for each tower
  for (int i = 0; i < 3; i ++){
    if (!tower[i].isEmpty()){
      disc[i] = tower[i].peek();
    }
    else{
      disc[i] = new Disc(size);
    }
  }

  //To the disc follows the mouse after it is mousePressed
```

```
  for (int i =0; i < 3; i ++){
    if (mouseX >= disc[i].xPos && mouseX <= disc[i].xPos + disc[i].Width &&
        mouseY >= disc[i].yPos && mouseY <= disc[i].yPos + disc[i].Height) {
      move = -move;
      referencedDisc = i;
      d = tower[referencedDisc].peek();
    }
  }

  //Repositions the disc after a click in a allowed local
  if (mouseX >= d.xPos && mouseX <= d.xPos + d.Width &&
      mouseY >= d.yPos && mouseY <= d.yPos + d.Height) {
    for (int i = 0; i < 3; i ++){
      if (move == -1 && mouseX >= pillar[i].xPos -30 && mouseX <= pillar[i].xPos +
          mouseY >= pillar[i].yPos && mouseY <= pillar[i].yPos + pillar[i].Height &
        tower[referencedDisc].pop();
        d.xPos = pillar[i].xPos - d.Width/2 + 5;
        d.yPos = height - (d.Height * (tower[i].size() + 1)) - 10;
        disc[i] = tower[i].push(d);
        referencedDisc = i;
      }
    }
  }
}
```

O código abaixo do comentário *//Repostions the disc after a click in a allowed local* é funadamental para todo o programa, pois é ele que administra o algoritimo que possibilita a movimentação de um disco de torre para outra, adminstrando os stacks envolvidos. Em grau de importância, ele está em primeiro lugar. Nesse código, basicamente é retirado o disco da torre de que saiu, setada sua posição x para o meio da torre escolhido para ser colocado e sua posição y para acima do disco mais superior da torre escolhida, adicionado o disco à torre que foi colocado e atualizado o referencedDisc para o código ser executado corretamente.

2.4.7 void changeTotalDiscs()

Chamada dentro de mousePressed(). Gere o total de discos e os botões para aumentar ou diminuí-los. Aparece apenas no menu inicial.

2.4.8 void startGame()

Chamada dentro de mousePressed(). Gere o botão de Play e causa o subsequente iniciar do jogo após ele ser clicado. Aparece apenas no menu inicial.

2.4.9 void mousePressed()

Proporciona que moveDiscs(), changeTotalDiscs() e startGame() sejam chamadas apenas quando o mouse seja clicado. Vale a pena lembrar que essas funções também tem internamente condições para poderem ser executadas.

3 Resultados

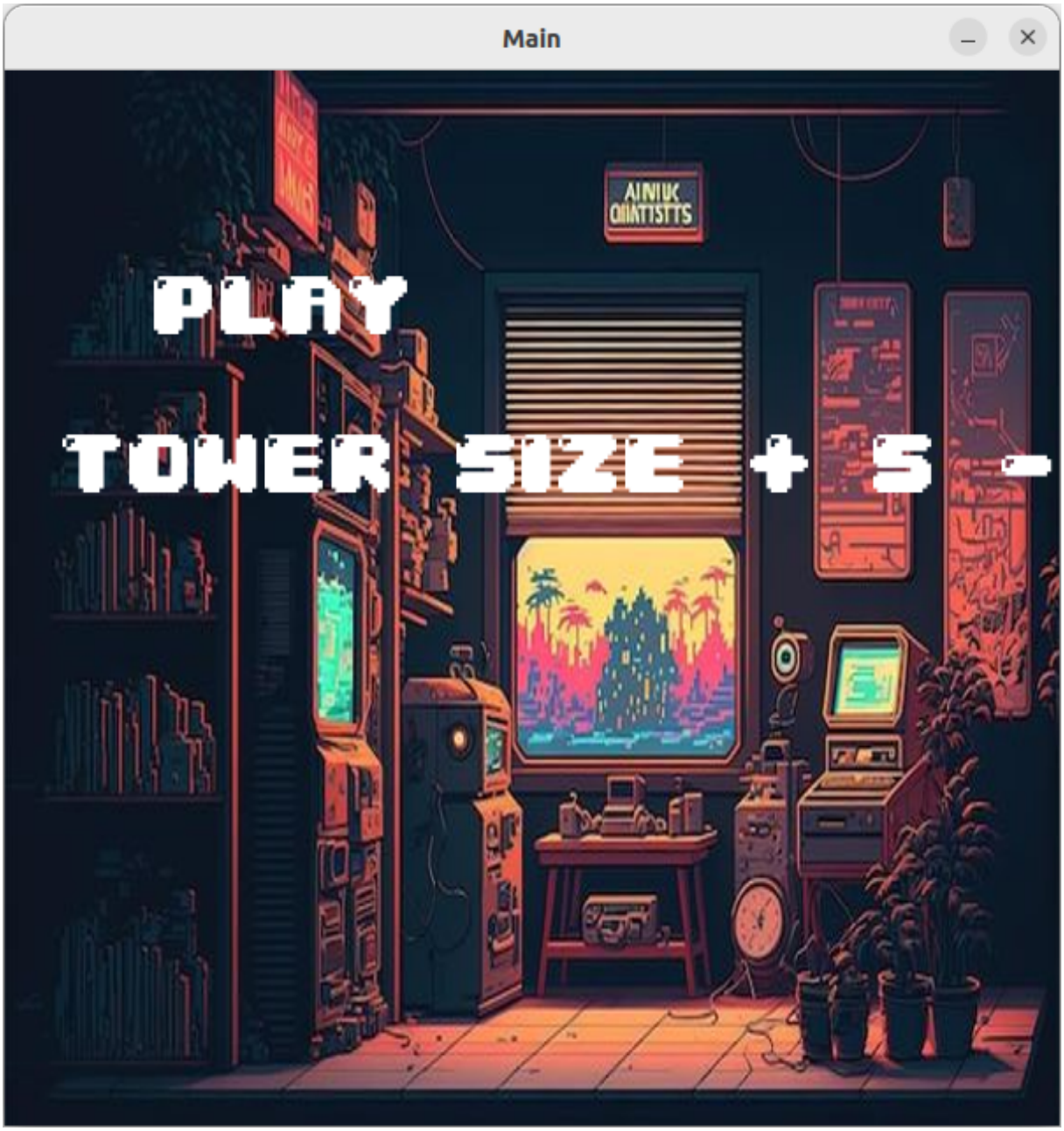

Figura 1 – Menu Inicial

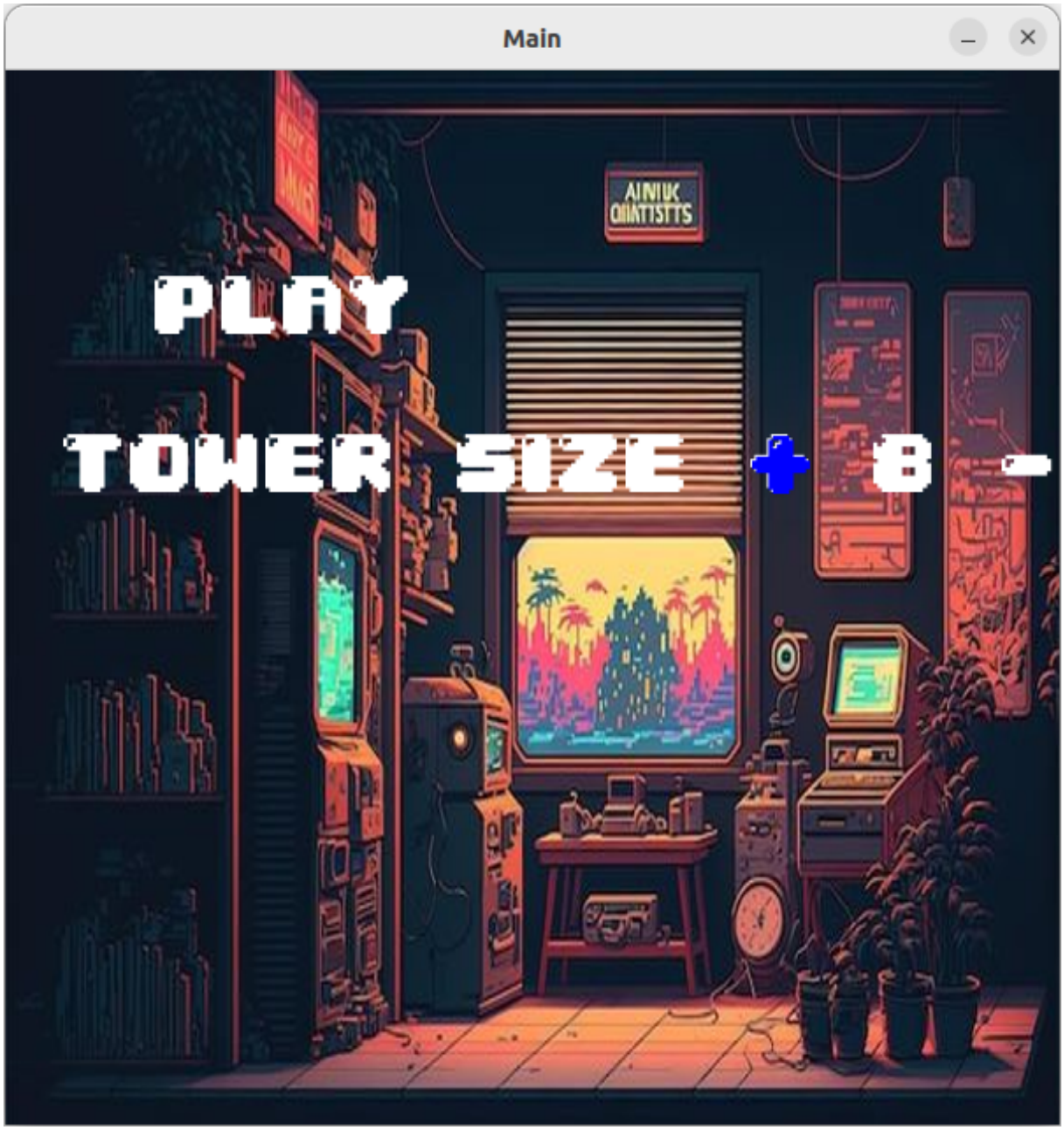

Figura 2 – Botão para aumentar tamanho da torre funcionando

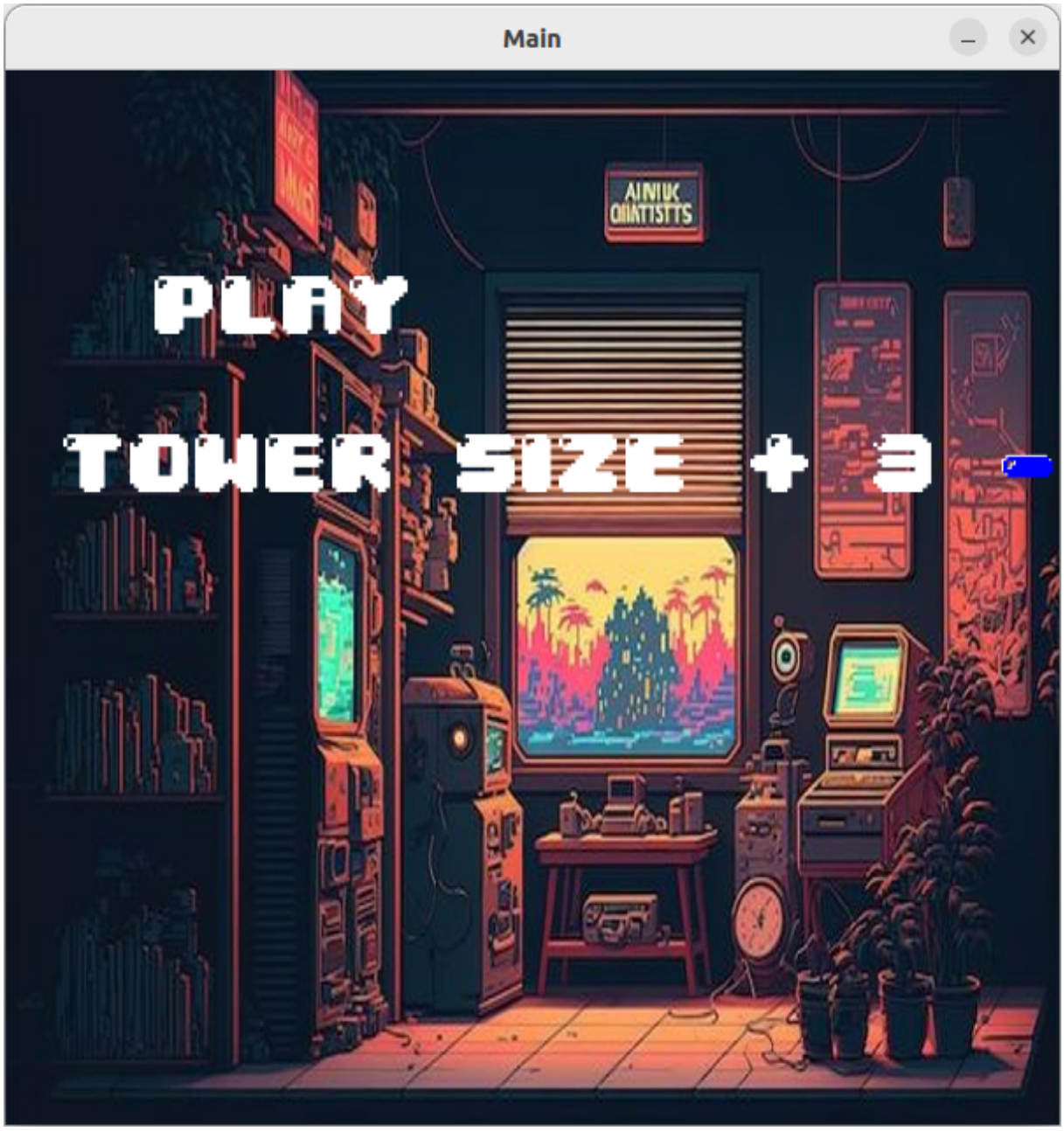

Figura 3 – Botão para diminuir tamanho da torre funcionando

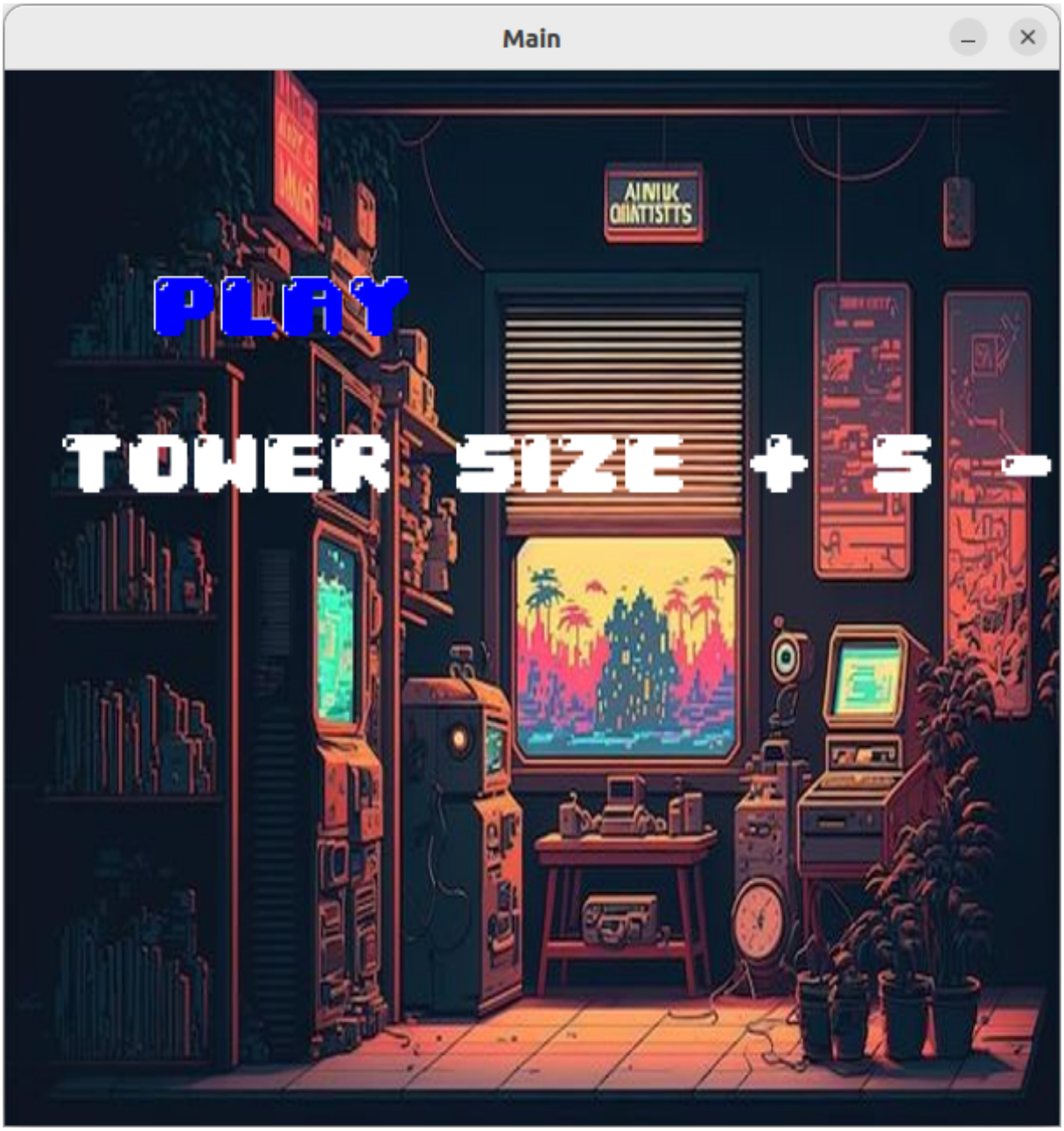

Figura 4 – Botão play funcionando

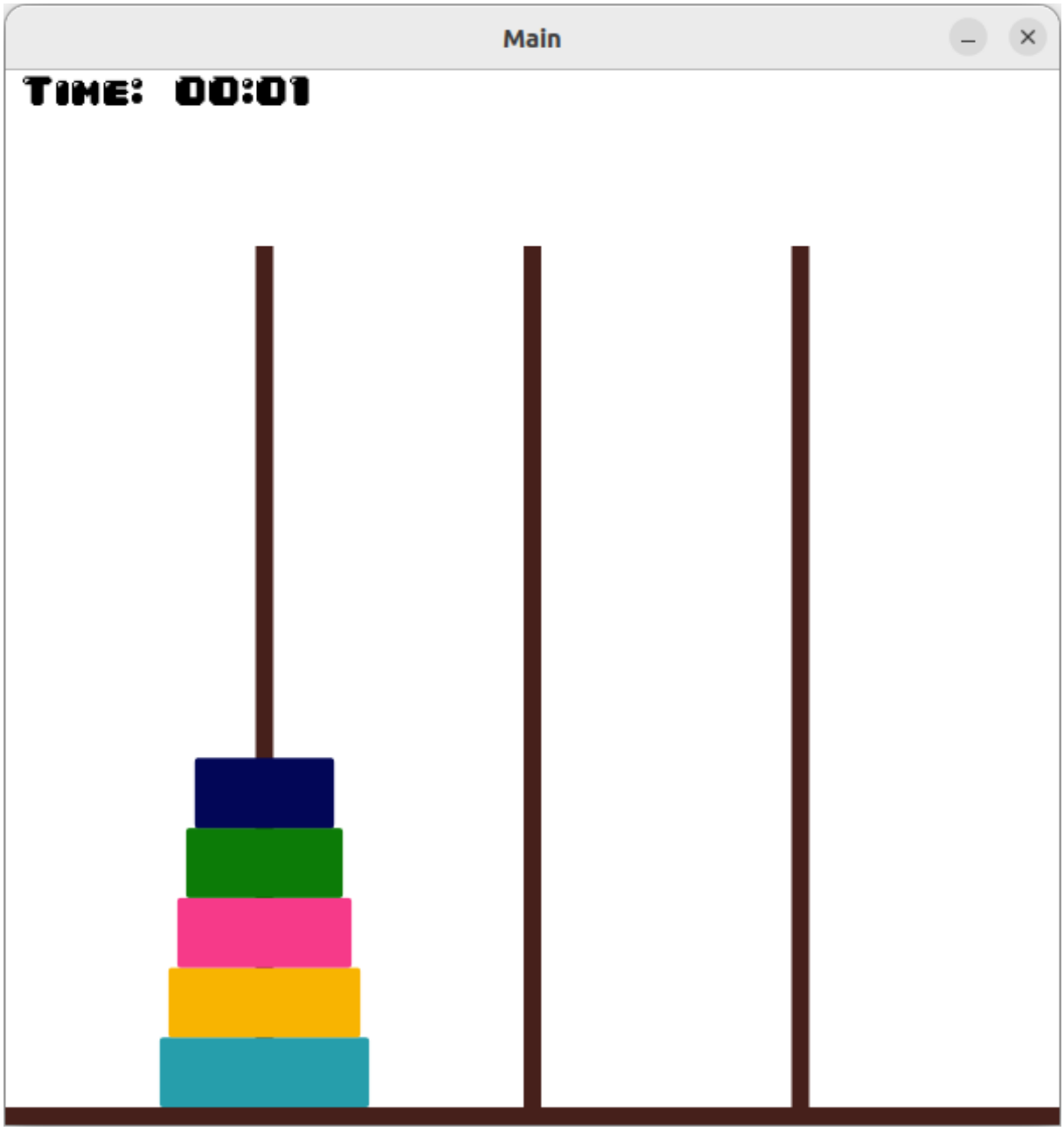

Figura 5 – Visão inicial do jogo

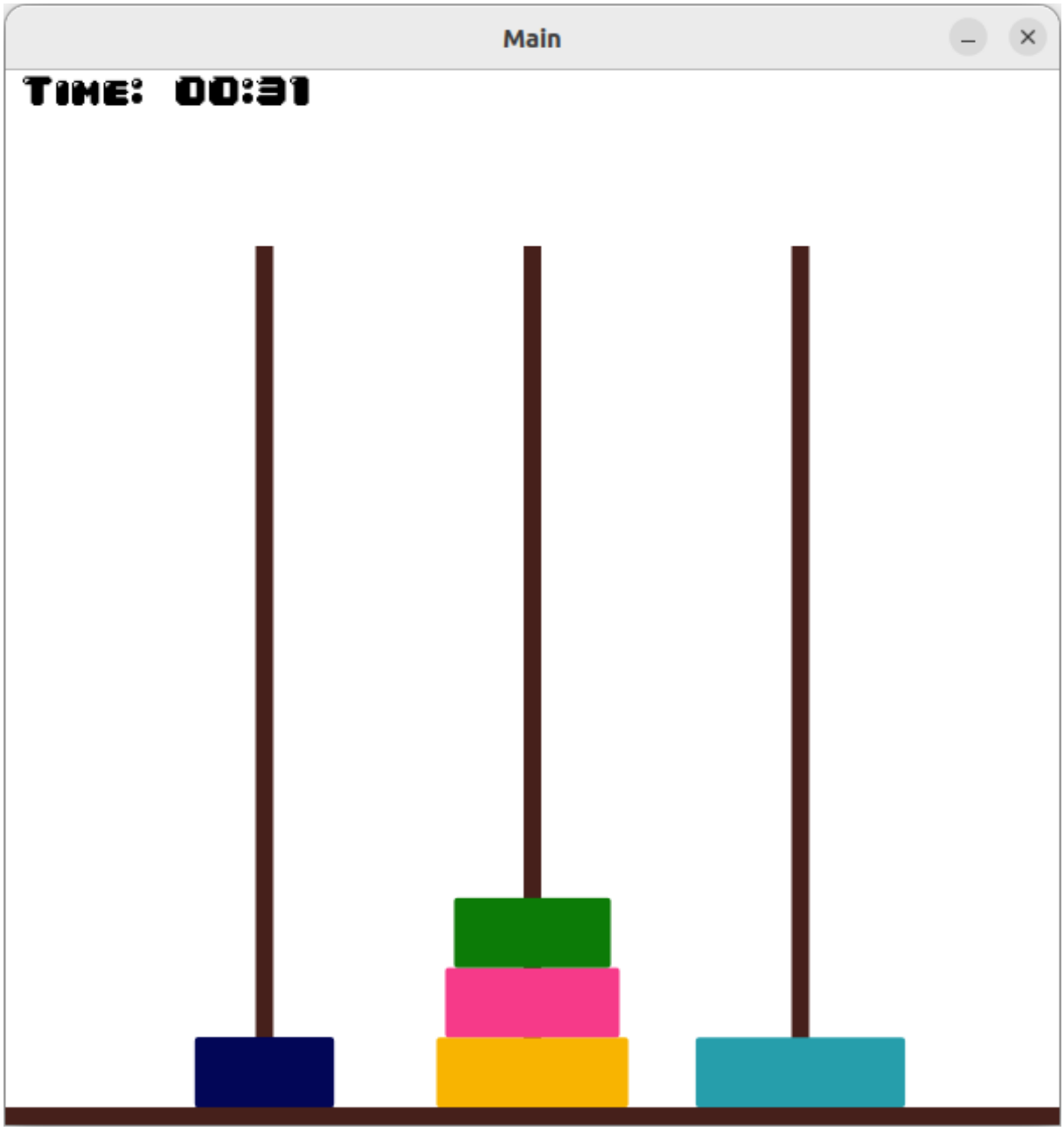

Figura 6 – Discos movimentados

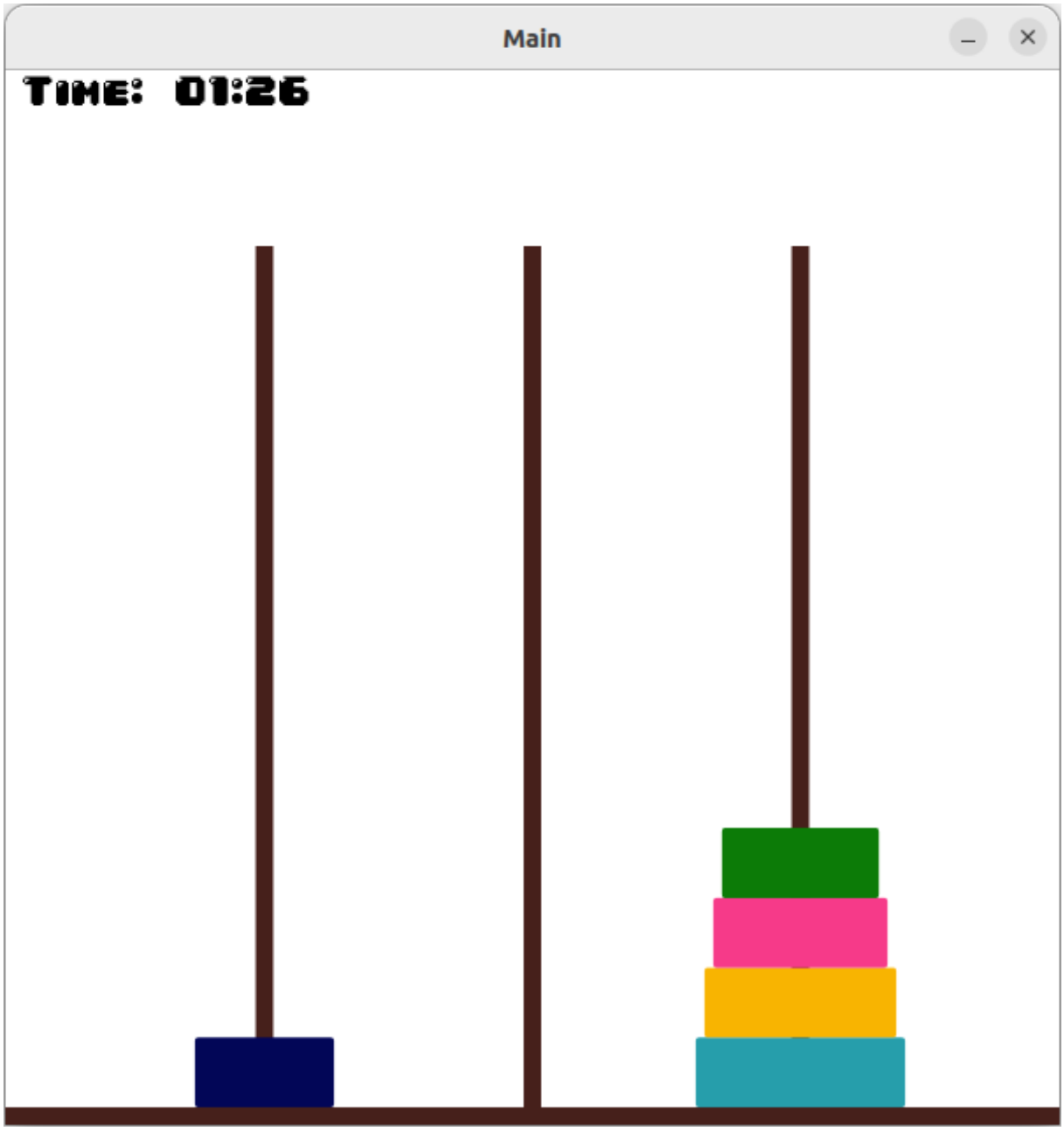

Figura 7 – Torre quase ordenada

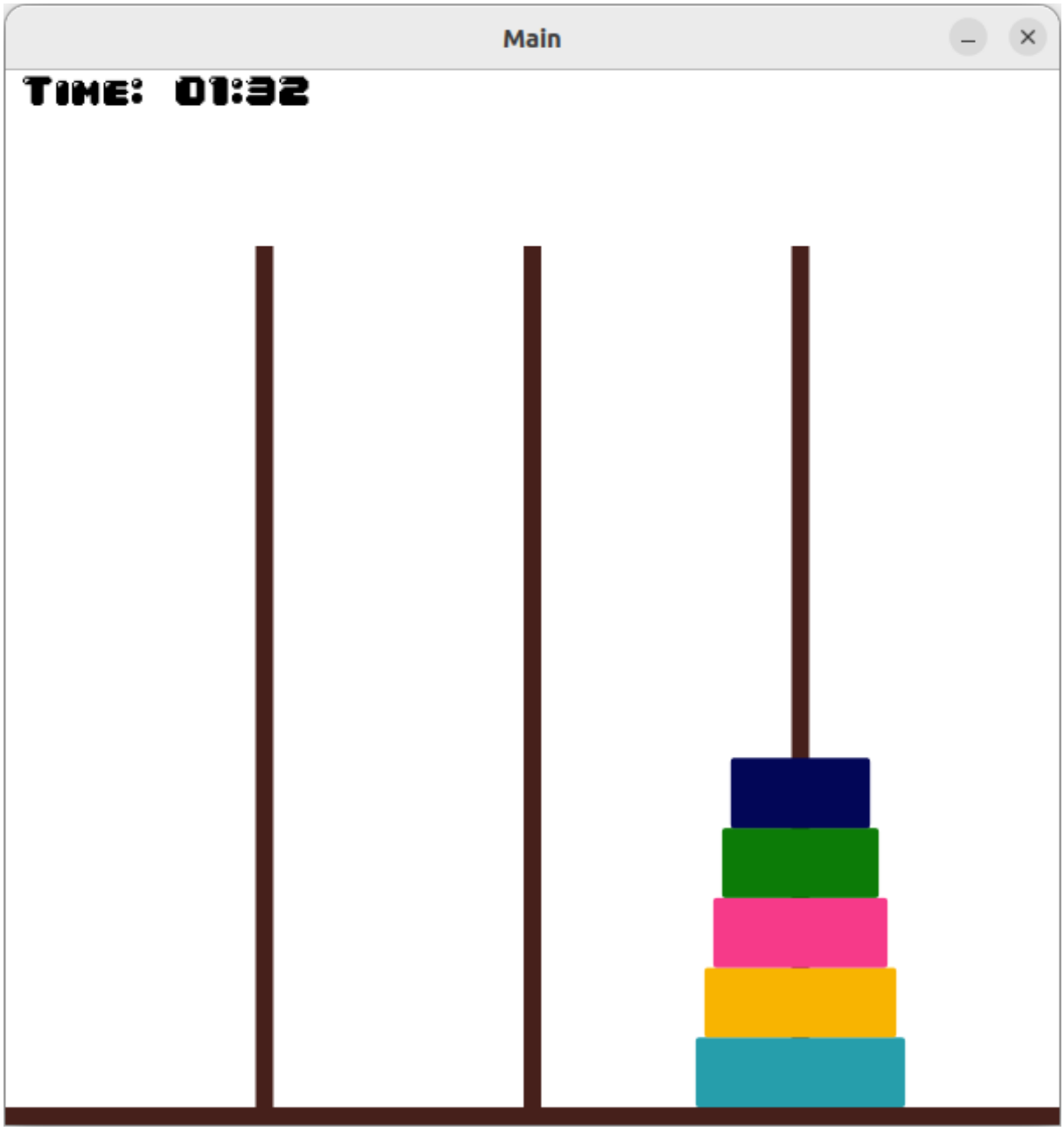

Figura 8 – Torre ordenada (um instante antes da aparição do menu final)

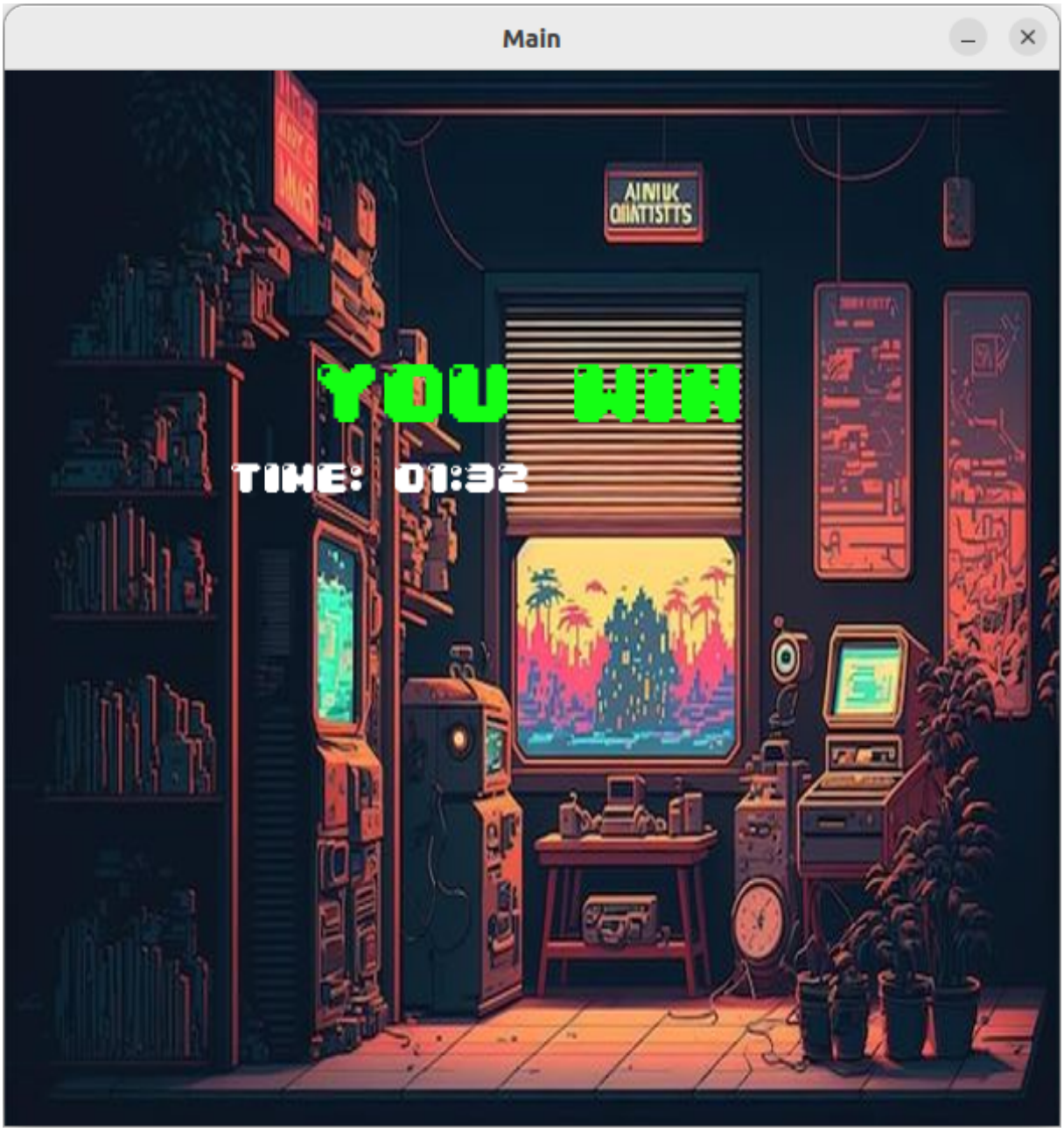

Figura 9 – Menu Final

4 Conclusão

- Todos os resultados estão de acordo com o esperado. O jogo foi criado corretamente implementando suas regras e stacks para armazenar os discos de cada torre. O aprendizado prático em stacks objetivado também foi alcançado.
- A principal dificuldade durante a execução do programa foi a criação da função *moveDiscs()* para gerenciar as torres e discos. Houveram diversas tentativas falhas até alcançar uma abstração que funcionasse corretamente. Foi revolucionário para o funcionamento

correto da função e código a criação de um index *referencedDisc* e as 3 instâncias de peek disc para cada torre. Outra dificuldade digna de ser mencionada foi criar o construtor de Disc planejado para uso dentro de um loop for, pois os cálculos das dimensões e posições do disco se tornaram muito complexos, por falta de planejamento e má implementação, principalmente.

- Diante da realização desse experimento os maiores aprendizados foram da necessidade de se planejar um código antes de sua criação - o que teria evitado a dificuldade em criar *moveDiscs()* e os calcúlos do construtor Disc -, o do uso máximo de 1 hora e 30 minutos consecutivos para períodos focados de trabalho (após isso tudo se torna confuso e a irritabilidade aumenta) e a melhora na capacidade de abstração e uso de stacks.