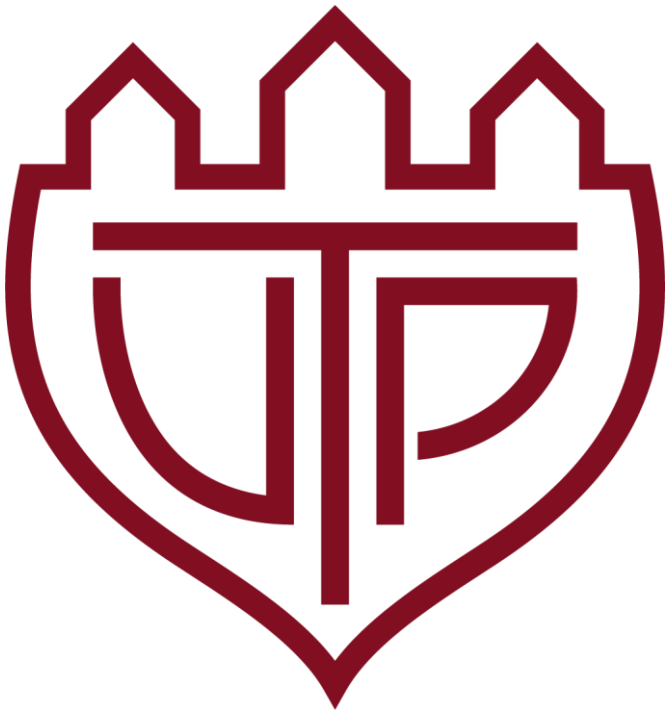


Programowanie aplikacji mobilnych

Tapper

Mateusz Tokarczyk, Jakub Olszak

Informatyka Stosowana, semestr VI 2019/2020

https://github.com/mati1000500100900/Tapper-Processing

Spis treści

1. Opis gry

Tapper jest grą o prostej mechanice, polegającą na obsługiwaniu klientów baru poprzez podawanie im piw po ladzie. Musimy obsłużyć każdego klienta aby ukończyć dany poziom. Każdy poziom składa się z 4 lad rozmieszczonych w różny sposób, przy których pojawiają się klienci. Aby obsłużyć danego klienta należy mu dostarczyć piwo. Trzeba jednak uważać aby nie „rzucić" piwa na ladę, przy której nie mamy żadnego klienta, ponieważ w takim przypadku kufel się zbije a my stracimy życie (strata życia jest równoznaczna z rozpoczęciem poziomu od nowa). Niektórzy klienci oddają kufel z powrotem i należy go złapać aby on się nie rozbił i żebyśmy nie przegrali.

2. Mechanika gry

W grze steruje się za pomocą gestów, gest przesunięcia w górę oraz w dół odpowiada za przemieszczanie się odpowiednio między stołami. Natomiast gest w lewo lub w prawo odpowiada za wydanie piwa klientowi. Klient po odebraniu piwa ma $\frac{1}{8}$ szansy na postanowienie wypicia piwa na miejscu. W takim przypadku klient zatrzymuje się na chwilę oraz następnie odsyła pusty kufel z piwem w stronę gracza. Gracz przechodzi kolejno przez poziomy mając do dyspozycji 5 żyć. Życie może stracić na 3 sposoby:

- Klient dojdzie do końca lady i nie zostanie obsłużony,
- Puszczenie kufla na stół, przy którym nie ma żadnego gościa,
- Nie złapanie kufla przez klienta, który postanowił wypić piwo na miejscu

Gracz przechodzi kolejno przez 13 poziomów. Po przejściu ostatniego wracamy do pierwszego poziomu i tak do utraty wszystkich żyć.

Gra oferuje możliwość pauzy w dowolnym momencie. Wykonywana ona jest poprzez gest „Wstecz" na Androidzie.

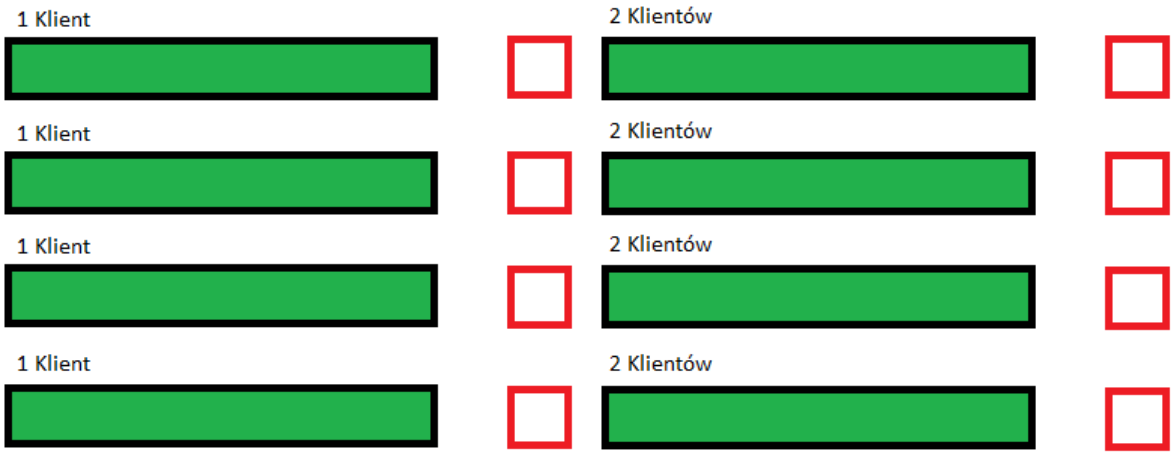
3. Układ poziomów

Gra składa się z czterech etapów:

- Saloon (2 poziomy),
- Sports Bar (3 poziomy),
- Punk Bar (4 poziomy),
- Alien Bar (4 poziomy)

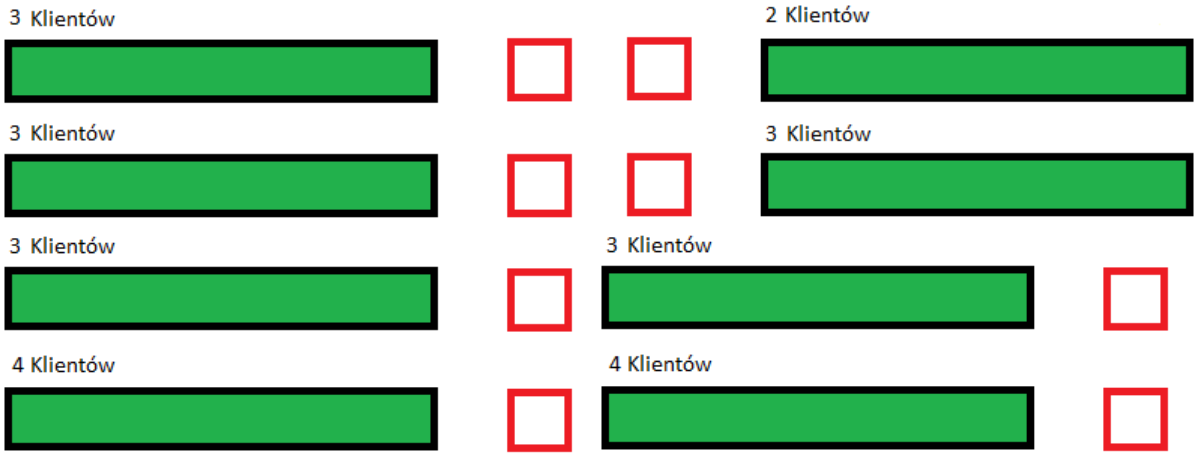
Poniżej przedstawione są układy stołów dla poszczególnych poziomów, wraz z ilością gości pojawiającą się przy każdym z nich:

Poziom 1 oraz poziom 2:

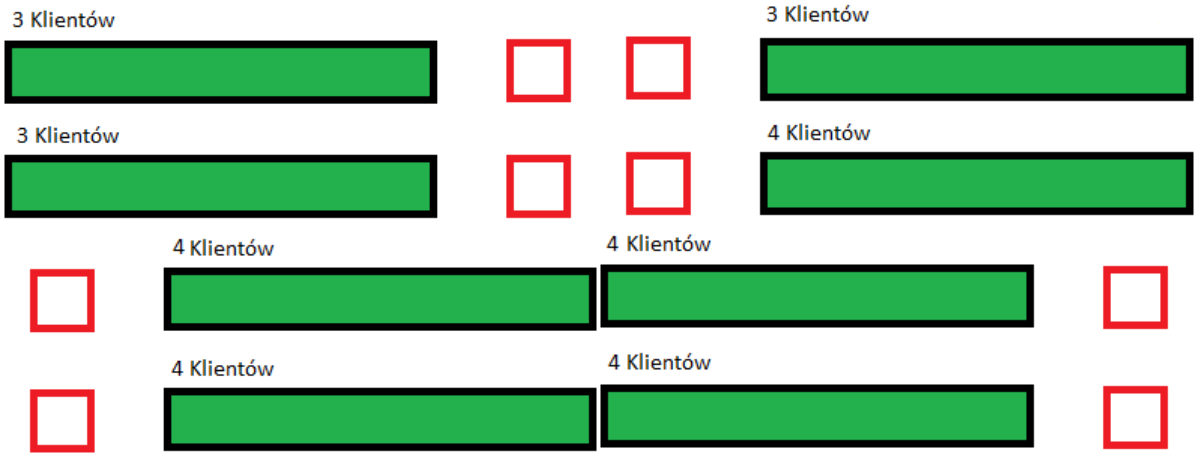


Poziom 3 oraz poziom 4:

Poziom 5 oraz poziom 6:

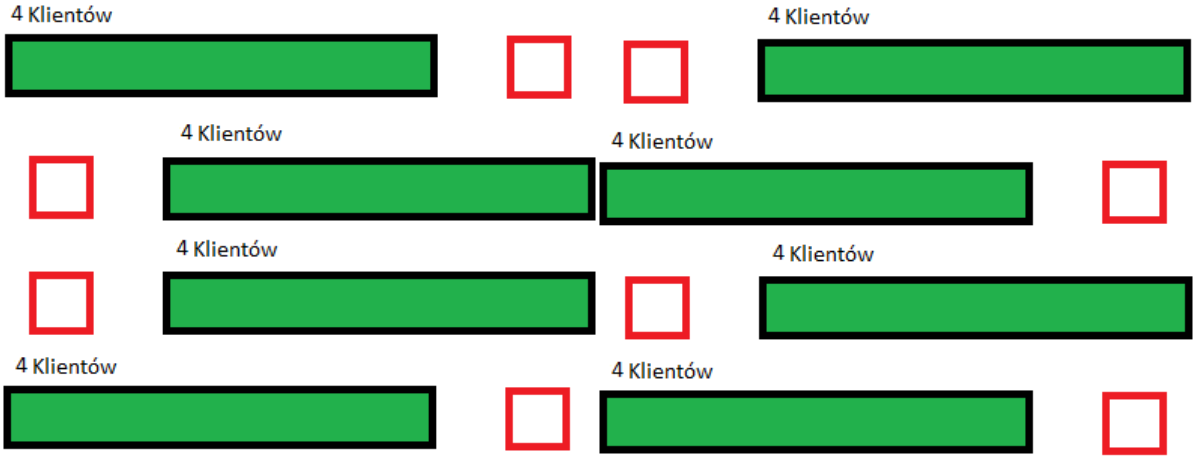


Poziom 7 oraz poziom 8:



Poziom 9 oraz poziom 10:

Poziom 11 oraz poziom 12:



Poziom 13:

4. Implementacja

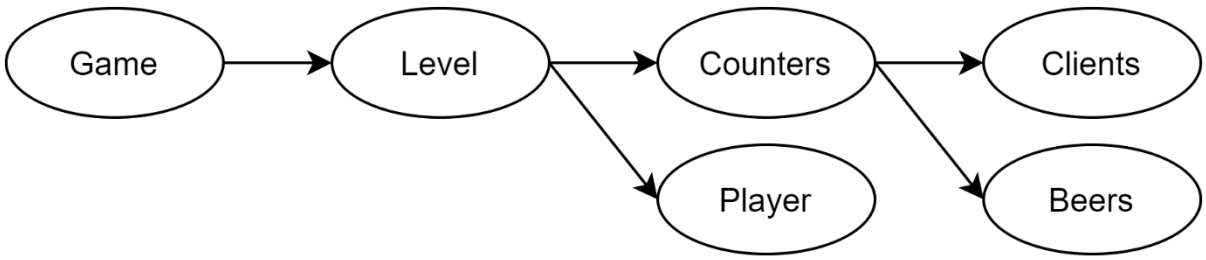
Projekt został napisany przy użyciu Processing (https://processing.org/). Jest to biblioteka języka Java do tworzenia aplikacji na canvasach. Do tworzenia grafik potrzebnych do gry został wykorzystany GIMP.

Aplikacja została zaprojektowana w oparciu o wzorzec chain of responsibility.

Łańcuch odpowiedzialności (nazywany też łańcuchem zobowiązań) jest czynnościowym wzorcem projektowym, który pozwala na oddzielenie nadawcy żądania od obiektu, który je zrealizuje. Osiąga to poprzez utworzenie łańcucha z obiektów, potencjalnie mogących obsłużyć żądanie.

https://bulldogjob.pl/news/494-wzorzec-projektowy-lancuch-zobowiazan

Diagram klas na których oparta jest główna mechanika gry:



Klasa Gry (Game.pde):

```
class Game {
  ArrayList<Level> levels;
  int lives;
  int score;
  int currentLevel;
  Game() {
    this.newGame();
  }
  void nextLevel() {
    currentLevel=(currentLevel+1)%13;
    currentScene=levels.get(currentLevel);
    levels.get(currentLevel).restart();
  }
```

```
  void newGame() {
    lives=5;
    score=0;
    currentLevel=0;

    levels = new ArrayList<Level>();
    levels.add(new Level(true, 1, true, 1, true, 1, true, 1, 50, 0));
    levels.add(new Level(true, 2, true, 2, true, 2, true, 2, 50, 1));
    levels.add(new Level(false, 2, false, 3, false, 3, false, 3, 75, 2));
    levels.add(new Level(false, 2, false, 3, false, 3, false, 4, 75, 3));
    levels.add(new Level(false, 3, false, 3, false, 3, false, 4, 75, 4));
    levels.add(new Level(true, 2, true, 3, false, 3, false, 4, 100, 5));
    levels.add(new Level(false, 3, false, 3, true, 4, true, 4, 100, 6));
    levels.add(new Level(true, 3, true, 4, false, 4, false, 4, 100, 7));
    levels.add(new Level(false, 4, false, 4, true, 4, true, 4, 100, 8));
    levels.add(new Level(true, 4, false, 4, false, 4, true, 4, 150, 9));
    levels.add(new Level(false, 4, true, 4, true, 4, false, 4, 150, 10));
    levels.add(new Level(false, 4, true, 4, false, 4, true, 4, 150, 11));
    levels.add(new Level(true, 4, false, 4, true, 4, false, 4, 150, 12));
  }
}
```

Klasa ta zawiera tablicę poziomów oraz metodę, która odpowiada za przełączanie kolejnych poziomów. Tutaj zdefiniowany jest układ, ilość klientów oraz punkty za obsługę.

Klasa Poziomu (Level.pde):

```
class Level extends Scene {
  ArrayList<Counter> counters;
  int number, customerPoints;
  Player player;
  float scales[]={0.927, 0.95, 0.975, 1};

  Level(boolean l1, int c1, boolean l2, int c2, boolean l3, int c3, boolean
l4, int c4, int customerPoints, int number) {
    player=new Player();
    this.number=number;
    this.customerPoints=customerPoints;
    freezed=30;
    counters=new ArrayList<Counter>();
    counters.add(new Counter(l1, c1));
    counters.add(new Counter(l2, c2));
    counters.add(new Counter(l3, c3));
    counters.add(new Counter(l4, c4));

    buttons.add(new Button(width-150, height-150, height/10, height/10,
">", "nextLevel"));
  }
  void draw() {
    if (paused) {
      image(pause, (width-(height*4)/3)/2, 0, 4*height/3, height);
    } else {
      pushMatrix(); // 4:3 Start
      translate((width-(height*4)/3)/2, 0);

      image(backgrounds[number], 0, 0, 4*height/3, height);

      pushMatrix();
      translate(0, -height/27);
      for (int i=0; i<counters.size(); i++) {
        Counter c=counters.get(i);
```

```
        translate(0, height/4.2);
        c.draw(scales[i], number);
        if (i==player.position) {
          c.drawPlayer(player, scales[i]);
        } else {
          c.drawTap(player, scales[i]);
        }
      }
      popMatrix();
      image(doors[number], 0, 0, 4*height/3, height);

      image(scoreboard, 0, 0, 4*height/3, height);
      for (int i=0; i<5; i++) {
        if (game.lives-i>0) image(beer, 2*height/3+height/75+(i*height/20),
height/80, height/20, height/20);
        else image(emptybeer, 2*height/3+height/75+(i*height/20),
height/80, height/20, height/20);
      }
      int scoreCache=game.score;
      for (int i=0; i<6; i++) {
        image(numbers[scoreCache%10],2*height/3-height/75-(i*height/24),
height/70, -height/22, height/22);
        scoreCache/=10;
      }

      popMatrix(); // 4:3 Stop
      if (freezed<=0) {
        frame++;
        for (int i=0; i<counters.size(); i++) {
          Counter c=counters.get(i);
          c.update(player, i);
        }
      }
    }
    this.checkForLose();
    if (this.checkForWin()) {
      //wygrane
      game.score+=1000;
      game.nextLevel();
    }
    if (freezed>0)freezed--;
  }


  void handleInputs(String type, int x, int y) {
    if (player.busy==0 && freezed==0) {
      if (type.equals("UP")) {
        player.decreasePosition();
      } else if (type.equals("DOWN")) {
        player.increasePosition();
      } else {
        counters.get(player.position).throwBeer(type, 10, player);
      }
    }
  }

  void restart() {
    frame=0;
    freezed=50;
    for (Counter c : counters) {
      c.restart();
```

```
    }
  }
  void checkForLose() {
    for (int i = 0; i<4; i++) {
      Counter c= counters.get(i);
      if (c.checkForLose(player, i)) {
        game.lives--;
        if (game.lives==0) {
          currentScene=new GameOver();
        } else this.restart();
      }
    }
  }
  boolean checkForWin() {
    boolean r=true;
    for (Counter c : counters) {
      if (c.customers.size()!=0 || c.beers.size()!=0) return false;
    }
    return r;
  }

  void pause() {
    paused=true;
    buttons.add(new Button(int(width/2-height/5.6), height/2+height/25,
int(height/2.8), height/15, "X", "returnToMenu"));
    buttons.add(new Button(int(width/2-height/5.6), height/2-height/15,
int(height/2.8), height/15, "II", "resumePause"));
  }
  void resumePause() {
    paused=false;
    buttons.remove(buttons.findByAction("returnToMenu"));
    buttons.remove(buttons.findByAction("resumePause"));
  }
}
```

W tej klasie zdefiniowane są obiekty lady oraz gracza. Zawiera metodę rysującą cały poziom, metodę sprawdzającą warunki wygrania lub przegrania oraz obsługującą pauzę. Jedną z ważniejszych metod zawartych w tej klasie obsługa sterowania graczem.

Klasa Gracza (Player.pde)

```
class Player {
  int position;
  int busy;
  Player() {
    busy=0;
    position=3;
  }
  void draw(float scaleX) {
    fill(255, 0, 0);
    stroke(0);
    if (busy<=0) image(idle[(frameCount/8)%2], ((4*height/3)-
height/6.3)*scaleX, -height/6, height/8, height/4);
    else {
      busy--;
      image(filling[9-((busy)%10)], ((4*height/3)-height/6.3)*scaleX, -
height/6, height/8, height/4);
    }
  }
```

```
  void drawTap(float scaleX) {
    image(tap, ((4*height/3)-height/6.3)*scaleX, -height/6, height/8,
height/4);
  }

  void decreasePosition() {
    position=(position+3)%4;
  }

  void increasePosition() {
    position=(position+1)%4;
  }
```

Klasa gracza rysuje odpowiednio obiekt gracza na planszy.

Klasa Lady (Counter.pde)

```
class Counter {
  boolean fromLeft;
  int customersCount;
  int randomOffset;
  int beerDelay;
  ArrayList<Customer> customers;
  ArrayList<Beer> beers;

  Counter(boolean l, int cc) {
    this.beerDelay=0;
    this.fromLeft=l;
    this.customersCount=cc;
    this.randomOffset=int(random(0, 10));
    this.restart();
  void restart() {
    customers = new ArrayList<Customer>();
    beers = new ArrayList<Beer>();
    for (int i=0; i<customersCount; i++) {
      customers.add(new Customer(floor(random(10)), (-30*i)+randomOffset));
    }
  void draw(float scaleX, int number) {
    fill(100);
    stroke(0);
    pushMatrix();
    if (!fromLeft) {
      translate(4*height/3, 0);
      scale(-1, 1);
    }
    scale(scaleX, 1);
    image(counter[number], 0, 0, ((height*4)/3)*0.9, height/12);
    for (Customer c : customers) {
      c.draw(number);
    }
    for (Beer b : beers) {
      b.draw();

    popMatrix();
  }
  void update(Player player, int number) {
    if (beerDelay>0) {
      if (beerDelay==1) beers.add(new Beer());
      beerDelay--;
    }

    for (Customer c : customers) {
      c.update();
```

```
  }
  for (Beer b : beers) {
    b.update();
  }
  for (Customer c : customers) {
    for (Beer b : beers) {
      if (c.x>=b.x) { // client colided with beer
        if (b.full && c.drinking==0 && !c.isGoingBack) {

          if (int(random(0, 8))==2) {
            c.drinking=30;
            b.full=false;
            b.wait=30;
          } else {
            beers.remove(b);
            c.isGoingBack=true;
          }
          break;
  for (Customer c : customers) { // garbage collection
    if (c.x<0 && c.isGoingBack) {
      game.score+=game.levels.get(game.currentLevel).customerPoints;
      customers.remove(c);
      break;
    }
  }
  for (Beer b : beers) {
    if (b.x>320 && player.position==number && !b.full || b.x>340) {
      game.score+=100;
      beers.remove(b);
      break;
    }
void drawPlayer(Player p, float scaleX) {
  pushMatrix();
  if (!fromLeft) {
    translate(4*height/3, 0);
    scale(-1, 1);
  }
  p.draw(scaleX);
  popMatrix();

void drawTap(Player p, float scaleX) {
  pushMatrix();
  if (!fromLeft) {
    translate(4*height/3, 0);
    scale(-1, 1);
  }
  p.drawTap(scaleX);
  popMatrix();

boolean checkForLose(Player player, int pos) {
  boolean r=false;
  for (Customer c : customers) {
    if (c.checkForLose()) {
      return true;
    }
  }
  for (Beer b : beers) {
    if (b.checkForLose(player, pos)) {
      return true;
    }
  }
```

```
    return r;
  }
  void throwBeer(String type, int delay, Player player) {
    if ((fromLeft && type.equals("LEFT")) || (!fromLeft &&
type.equals("RIGHT"))) {
      beerDelay=delay;
      player.busy=delay;
```

Klasa lady zawiera klientów i piwa. Odpowiada za ich rysowanie. Klasa ta również obsługuje kolizje obiektów (obiekty klientów oraz piw) na danej ladzie. Ta klasa jest również kolejnym poziomem sprawdzania warunków przegrania oraz wygrania.

Klasa Klienta (Customer.pde)

```
class Customer {
  int x, drinking, index;
  boolean isGoingBack;

  Customer(int index, int x) {
    this.x=x;
    this.index=index;
    drinking=0;
    isGoingBack=false;
  }
  void draw(int number) {
    stroke(0);
    fill(64);
    if (x*height/300>0) {

      pushMatrix();
      translate(x*height/300, 0);
      if (isGoingBack){
        translate(height/15, 0);
        scale(-1, 1);
      }
      image(clients[number][index], 0, 0, height/15, -height/15);
      popMatrix();
    }
  }
  void update() {
    if (drinking>0) {
      drinking--;
      if (drinking==0) {
        isGoingBack=true;
      }
    } else {
      if (isGoingBack) {
        this.x-=3;
      } else {
        if (drinking==0) {
          this.x++;
        }
      }

  boolean checkForLose() {
    return (this.x>330);
```

Klasa klienta odpowiada za rysowanie obiektów klientów przypisanych do lad, oraz jest kolejnym poziomem sprawdzania warunków wygranej bądź przegranej.

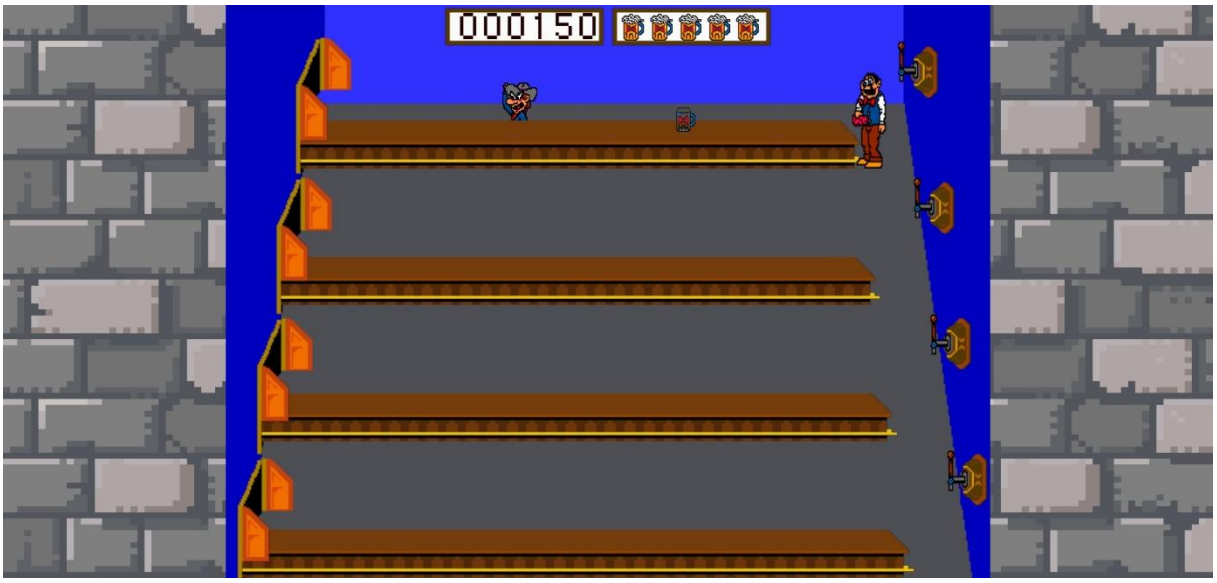
Klasa Piwa (Beer.pde)

```
class Beer {
  int x;
  boolean full;
  int wait;

  Beer() {
    wait=0;
    x=340;
    full=true;
  }
  void draw() {
    fill(255, 255, 0);
    stroke(0);
    if (x*height/300>0 && wait==0) {
      if (full) image(beer, x*height/300, height/50, height/20, -
height/20);
      else image(emptybeer, x*height/300, height/50, height/20, -
height/20);
    }
  }

  void update() {
    if (wait>0) {
      wait--;
    } else {
      if (full) x-=4;
      else x+=2;
    }
  }
  boolean checkForLose(Player player, int pos) {
    if (this.x<0 && this.full) return true;
    if (this.x>=339 && !this.full && player.position!=pos) return true;
    return false;
  }
}
```

Klasa ta odpowiada za rysowanie obiektu piwa oraz jest kolejnym etapem sprawdzania warunków przegrania lub wygrania.

5. Zrzuty ekranu

Etap I



Etap II

Etap III

Etap IV

Ekran menu

Ekran pauzy

Ekran końca gry

Ekran testu dotyku