МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ(№ 43)

КУРСОВАЯ РАБОТА   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Е.О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ «ОНЛАЙН-БРОНИРОВАНИЕ ТУРОВ» СРЕДСТВАМИ ЯЗЫКА UML |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**Содержание**

|  |  |
| --- | --- |
| 1. Задание…………………………………………….................................... | 3 |
| 2. Анализ предметной области………………………………….…………. | 3 |
| 3. Разработка программного обеспечения……………………………....... | 3 |
| 3.1. Разработка структуры иерархии классов…………….…………..… | 3 |
| 3.2. Проектирование интерфейса программы.……………………..….... | 4 |
| 3.3. Разработка модулей программного продукта ………..……………. | 5 |
| 4. Тестирование.…………...……………………………………..…...…….. | 6 |
| 5.Руководство пользователя………………………….…………………… | 10 |
| Заключение ………………………………………………………….……... | 10 |
|  |  |
| Приложение……………………………………………………………..….. | 11 |
|  |  |
|  |  |

1. **Задание**

Разработать программный продукт представляющий собой игру со следующим содержанием.

«Великий колдун сказал что шадокские ракеты терпят неудачи потому, что не хватает транзисторов в системах безопасности. Транзисторы нужного типа собирают с растений, произрастающих на огородах Гибби. Вы должны отправиться на планету Гибби и привести нужные транзисторы. Будьте осторожны, Гибби очень умны. Они позволяют Шадокам забираться в свои огороды, но защищают свои посадки. Всякий раз как, только растение зацветает, они мчатся, чтобы собрать урожай раньше Шадока».

1. **Анализ предметной области**

Все действия игры происходящие в огороде Гибби будут отображаться на экране пользователя. Отвечать за параметры экрана и загрузку внешних файлов декораций будет класс «Screen». Из действующих лиц в игре фигурируют: гибби, игрок, растения. Каждый из которых, в силу значительного количества отличий друг от друга будет представлен своим классом «Gibby», «Player», «Decoration».

Дополнительно внимания заслуживает отделение логики передвижения игрока, логику взаимодействия пользователя с клавиатурой и конвертацию данных в координаты игрока на экране стоит вынести в отдельный класс «ActionController».

1. **Разработка программного обеспечения**
   1. **Разработка структуры иерархии классов**

За проецирование всего происходящего в игре на экран, отвечает класс «Screen». В ходе своей инициализации, загружающий все необходимые для отображения игрового процесса файлы, такие как, изображения: игрока, гибби, растений, заднего фона огорода.

Главной же задачей этого класса будет не загрузка вспомогательных изображений а отображение происходящего в игре на экране пользователя. За это будет отвечать метод : *Screen.draw().* Здесь для значительного облегчения процесса разработки, уместно использовать паттерн Строитель. Так как игровой процесс из-за его постоянной динамики можно сравнить с воспроизведением видео. А видео можно дискретно разбить на кадры.

Так и отображение каждого кадра игрового процесса требует некоторой обработки , такой как, обработка действий игрока-передвижения, посредством использования клавиатуры. Проверки на выход за размеры экрана. Обработка логики цветения растения и соответственно обработка статуса того или иного растения, проверка на совпадение координат игрока и гибби с координатами активного растения. Что делает процесс рисования кадра довольно сложным, однако шаблон строитель позволяет нам инкапсулироваться от всей этой логики и предоставить классу Screen самостоятельно разбираться с построением картинки, а нам лишь вызывать метод обновления экрана. Паттерн схематично представлен на рис.3.1.

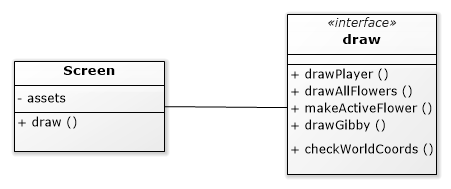


Рис.3.1.1

Несмотря на то, что растения в отличии от гибби и игрока не могут двигаться по игровому полю, у них можно выделить общие данные, такие как: координаты по оси Х и оси Y. И вынести их в общий класс «Actor», что позволит использовать порождающий паттерн Фабрика и инкапсулировать логику «актеров» от класса Screen, чтобы последний занимался лишь отображением объектов не вникая в детали реализации того или иного актера. Дополнительный же функционал передвижения будет реализован к классах «Gibby» и «Player» от наследованных от класcа «Actor».

Для обработки взаимодействия пользователя и внедрения её в игровой процесс используется класс «ActionController». Реализующий собой паттерн мост для конвертации данных от нажатия клавиши в положение игрока на экране. Принцип взаимодействия представлен на рис.3.2.

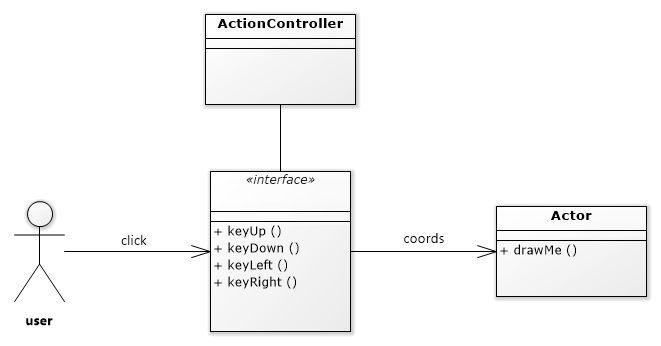


Рис.3.1.2

* 1. **Проектирование интерфейса программы**

Первым пользователю будет представлен экран с возможностью выбора, ознакомления с правилами или началом игры. Откуда у пользователя будет возможность вернуться на стартовый экран и начать игровой процесс, по завершении которого пользователь опять же будет перенаправлен на стартовый экран. Схема представлена на рис.3.2.

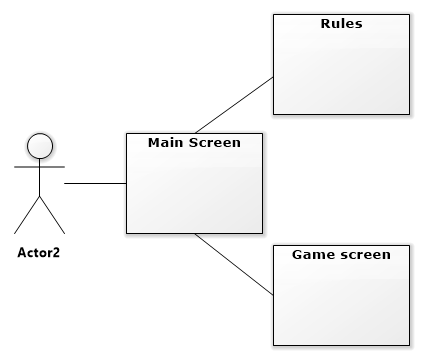


Рис.3.2

* 1. **Разработка модулей программного продукта**

Экраны для взаимодействия пользователя представляют собой 3 отдельных фаила, каждый из которых подключает к себе необходимые для отображения css файлы. И необходимые для реализации игровой логики js файлы. Загружаемые ресурсы, находятся в папке img. Схема на рис.3.3.1.

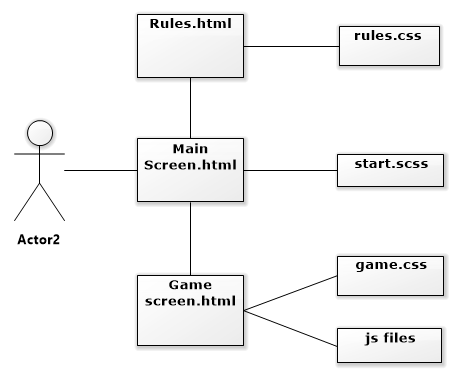


Рис.3.3.1

Иерархия расположения js файлов полностью совпадает с их иерархией в программной реализации. Каждый js файл представляет собой отдельный программный модуль.

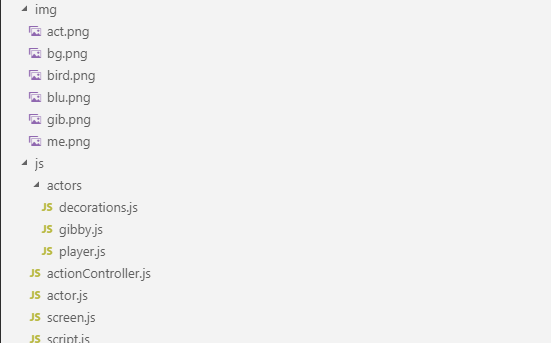
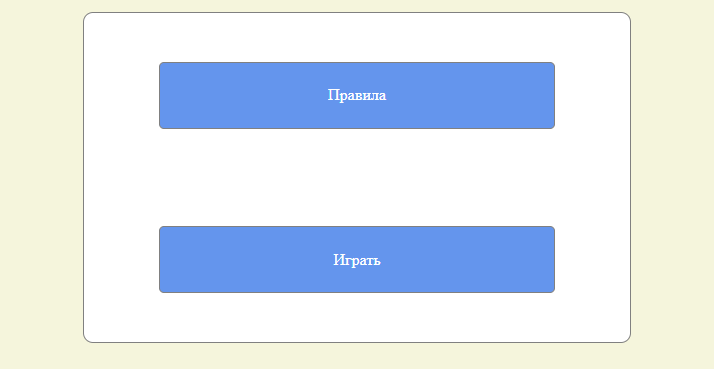
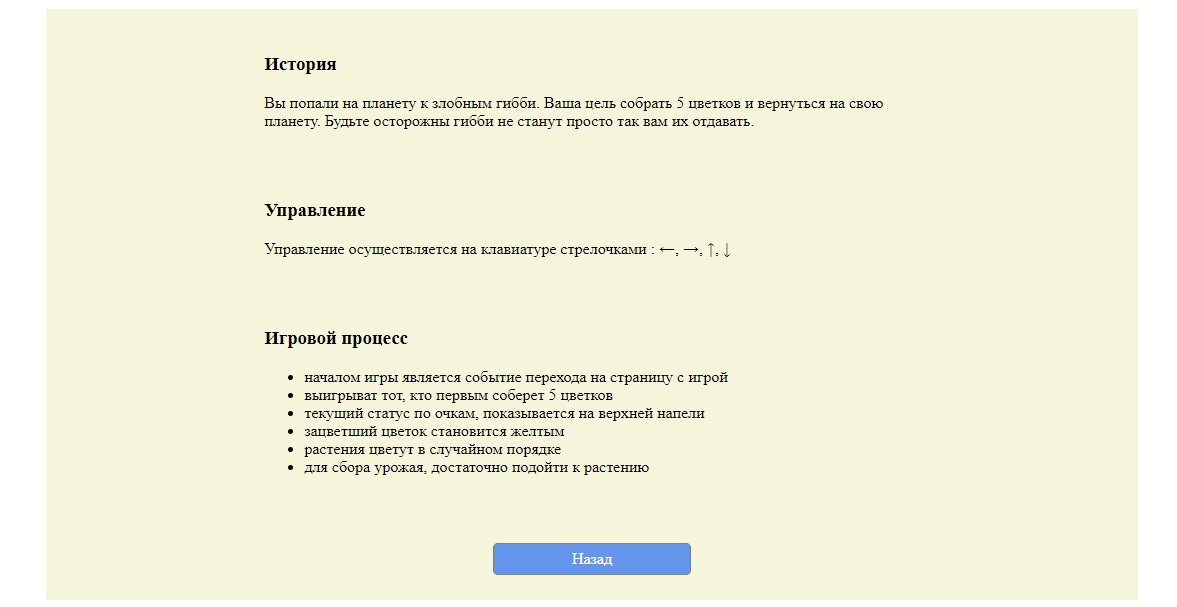


Рис.3.3.2

1. **Тестирование**
2. Стартовый экран



1. Экран с правилами



1. Игровой экран



1. Активное растение



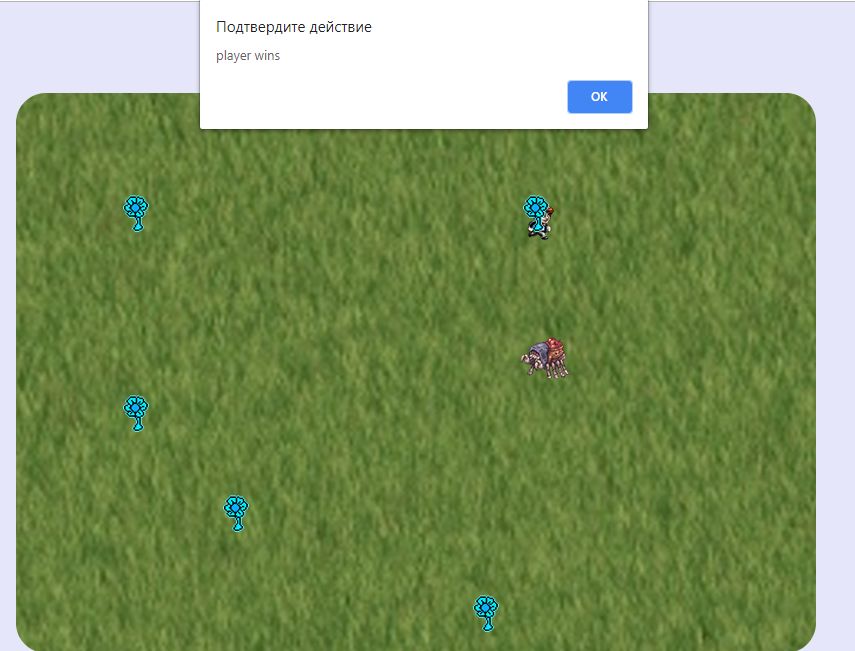
1. Передвижение игрока и Гибби



1. Сбор растения с обновлением игрового счетчика



1. Результат игры



1. **Руководство пользователя**

**Программные требования к игре:** Браузер Сhrome или Firefox (версия не младше 50)

Игра распространяется при помощи архива.

Для установки игры следует распаковать содержимое архива и открыть файл «start.html» при помощи браузера.   
 Управление осуществляется при помощи блока со стрелками на клавиатуре. Дополнительную информацию можно получить в блоке «Правила».

**Заключение**

В ходе выполнения курсового проекта было разработано приложение: «Игра Шадоки». Позволяющая пользователю посоревноваться в скорости со злобными гибби в процессе сбора зацветших растений.

В качестве среды выполнения используется Web-браузер. Для выполнения работы использовался язык разметки html, каскадные таблицы стилей css, для добавления возможности взаимодействия пользователя с web-страницей используется язык JavaScript.   
 В ходе написания программных модулей использовался Объектно-ориентированный подход, что позволило структурировать код и в значительной мере облегчило процесс разработки и поддержки программного продукта.  
 Так же были использованы такие паттерны, как строитель, фабрика и мост. Что позволило сделать код легко читаемым и понимаемым для других программистов, а так-же позволило избежать архитектурных ошибок в ходе разработки программного продукта.

**Библиографический список**

1. Рамбо, Якобсон, Буч: Введение в UML от создателей языка – ДМК-Пресс, 2015. – 496c.
2. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования – Диалектика, 2016. –736c.
3. М. Хавербеке. Выразительный JavaScript 2-е издание, 2015.-306с.
4. Томас А. Пауэл, Фриц Шнайдер. Полный справочник по JavaScript - ДМК-Пресс, 2016. – 700c
5. Пайлон, Питмен: UML 2 для программистов – Питер, 2012 – 240c.

**Приложение**

Класс «Screen»

class Screen {

setInitialData() {

var canvas = document.getElementById("myCanvas");

var ctx = canvas.getContext("2d");

var ballRadius = 10;

var flowerBorder = 20;

var me = new Image();

var gib = new Image();

var flow = new Image();

var actFlow = new Image();

var bg = new Image();

me.src = "img/me.png";

gib.src = "img/gib.png";

flow.src = "img/blu.png";

actFlow.src = "img/act.png";

bg.src = "img/bg.png";

return { canvas, ctx, ballRadius, flowerBorder,me ,gib, flow, actFlow, bg }

}

draw() {

if (!stop) {

ctx.clearRect(0, 0, canvas.width, canvas.height);

player.drawMe();

decoration.drawAllFlowers();

decoration.makeActiveFlower();

gibby.drawGibby();

player.checkMyCoords();

}

}

}

let ScreenItem = new Screen

Класс «Actor»

class Actor {

constructor(xCoord, yCoord) {

if (xCoord) {

this.xCoord = xCoord;

this.yCoord = yCoord;

}

this.xCoord = 0;

this.yCoord = 0;

}

get coords() {

const { xCoord, yCoord } = this

return { xCoord, yCoord }

}

set coords(xCoord, yCoord) {

this.xCoord = xCoord;

this.yCoord = yCoord;

}

catchFlower(givePointTo) {

flowX = flowY = false;

flowers[`fl${activeFlowerID}`]["color"] = "grey";

activeFlowerID = 0;

setActiveFlowerDelay = 2000;

let input = document.getElementById(givePointTo);

if (!input.value) {

input.value = 1;

} else {

console.log("input.value", input.value)

if (Number(input.value) + 1 == 5) {

setTimeout(() => {

alert(`${givePointTo} wins`);

window.location.href = "start.html";

stop = true;

}, 500)

}

input.value = Number(input.value) + 1;

}

}

};

Класс «ActionController»

class ActionControler {

constructor() {

}

keyRightHandler(e) {

if (e.keyCode == 39) {

rightPressed = true;

}

else if (e.keyCode == 37) {

leftPressed = true;

} else if (e.keyCode == 40) {

downPressed = true;

} else if (e.keyCode == 38) {

topPressed = true;

}

}

keyLeftUpHandler(e) {

if (e.keyCode == 39) {

rightPressed = false;

}

else if (e.keyCode == 37) {

leftPressed = false;

} else if (e.keyCode == 40) {

downPressed = false;

} else if (e.keyCode == 38) {

topPressed = false;

}

}

}

let controller = new ActionControler

Класс «Player»

class Player extends Actor {

drawMe() {

ctx.beginPath();

ctx.drawImage(bg, 0, 0);

ctx.closePath();

ctx.beginPath();

ctx.drawImage(me, meX, meY);

if (activeFlowerID && (meX > flowers[`fl${activeFlowerID}`]["flowX"] - ballRadius && meX < flowers[`fl${activeFlowerID}`]["flowX"] + flowerBorder) &&

(meY > flowers[`fl${activeFlowerID}`]["flowY"] - ballRadius && meY < flowers[`fl${activeFlowerID}`]["flowY"] + flowerBorder)) {

this.catchFlower("player")

}

ctx.fillStyle = "#0095DD";

ctx.fill();

ctx.closePath();

}

checkMyCoords() {

if (rightPressed && meX < canvas.width - ballRadius) {

meX += delta;

}

else if (leftPressed && meX > ballRadius) {

meX -= delta;

} else if (topPressed && meY > ballRadius) {

meY -= delta;

} else if (downPressed && meY < canvas.height - ballRadius) {

meY += delta;

}

}

}

let player = new Player

Класс «Gibby»

class Gibby extends Actor {

drawGibby() {

ctx.beginPath();

ctx.drawImage(gib, gx, gy);

ctx.fill();

ctx.closePath();

if (flowX && flowY) {

this.caclWay(gx, gy, flowX, flowY);

if (gx !== flowX) {

gx = this.makeStep(gx, flowX)

} else {

gy = this.makeStep(gy, flowY)

}

}

}

makeStep(g, flow) {

if (g < flow) {

if (flow - g < gDelta) {

g = flow

this.catchFlower("gibby");

} else {

g += gDelta;

}

} else {

if (g - flow < gDelta) {

g = flow

this.catchFlower("gibby");

} else {

g -= gDelta;

}

}

return g

}

caclWay(x, y, fx, fy) {

xWay = x - fx;

yWay = y - fy;

}

}

let gibby = new Gibby

Класс «Decoration»

class Decoration extends Actor {

drawAllFlowers() {

Object.keys(flowers).forEach(el => {

drawFlower(flowers[el].flowX, flowers[el].flowY, flowers[el].color);

})

function drawFlower(flowX, flowY, color) {

ctx.beginPath();

if (color == 'green') {

ctx.drawImage(actFlow, flowX, flowY);

} else {

ctx.drawImage(flow, flowX, flowY);

}

ctx.closePath();

}

}

randomInteger(min, max) {

var rand = min + Math.random() \* (max + 1 - min);

rand = Math.floor(rand);

return rand;

}

makeActiveFlower() {

if (setActiveFlowerDelay != 0) setActiveFlowerDelay -= 10;

let haveActive = 0;

Object.keys(flowers).forEach(el => {

if (flowers[el].color == "green") haveActive = 1;

});

if (!haveActive && !setActiveFlowerDelay) {

activeFlowerID = decoration.randomInteger(1, 5);

flowers[`fl${activeFlowerID}`].color = "green";

flowX = flowers[`fl${activeFlowerID}`]["flowX"];

flowY = flowers[`fl${activeFlowerID}`]["flowY"];

}

}

}

let decoration = new Decoration

Start.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Gamedev Canvas Workshop</title>

<link href="start.css" rel="stylesheet">

</head>

<body>

<div class="rules">

<div class="btnWrap">

<div class="btn">

<a href="rules.html">

<div>Правила

</div>

</a>

</div>

</div>

<div class="btnWrap">

<div class="btn">

<a href="game.html">

<div>Играть

</div>

</a>

</div>

</div>

</div>

</body>

</html>

Startt.css

body{

width: 100vw;

height: 100vh;

display: flex;

align-items: center;

justify-content: center;

background-color: beige;

}

.rules{

width: 40%;

height: 50%;

border: 1px solid grey;

border-radius: 10px;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

background-color: white;

}

.btnWrap{

flex:1;

width: 90%;

display: flex;

align-items: center;

justify-content:center;

}

.btn{

width: 80%;

height: 40%;

border: 1px solid grey;

border-radius: 5px;

text-align: center;

display: flex;

align-items: center;

justify-content: center;

background-color: cornflowerblue;

color: white;

}

a{

color:white;

text-decoration: none;

}