МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)

09.03.01 Информатика и вычислительная техника
Образовательная программа (профиль)
«Интеграция и программирование в САПР»
Кафедра «СМАРТ технологии»

ОТЧЕТ

По дисциплине «ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ»

«РАЗРАБОТКА ПО С ИСПОЛЬЗОВАНИЕМ THREE.JS»

“3D-игра Pacman-Three.js”

Состав команды: Хрусталев Г.Н. 181-326,
Зарубин А.Н. 201-323,
Козырь С.К. 201-323,
Селезнев В.К. 201-323,
Свинцицкий Р.Е. 201-324

Куратор проекта: зав. кафедрой «СМАРТ-технологии», к.т.н. Толстиков А. В.

Москва – 2021

АННОТАЦИЯ

Данный проект разрабатывается студентами 3-его и 1-ого курса групп 181-326, 201-323, 201-324, специальности «Информатика и вычислительная техника» образовательной программы «Интеграция и программирование в САПР», в составе объединения проектов «Лаборатории разработки САПР» и подпроекта «Центр САПР-разработки» по дисциплине «Проектная деятельность».

Отчетный документ включает в себя следующие разделы:

«Введение» — содержит описание проекта, поставленные задачи, основные этапы разработки проекта и достигнутые результаты проекта за прошлый семестр.

«Индивидуальные планы участников» — содержит список участников и распределение задач для каждого из студентов на время разработки проекта.

«Ход работы» — содержит полную информацию о этапах работы по проекту в этом семестре.

«Результаты» — содержит результаты выполнения поставленных задач, а также планы по дальнейшей доработке, улучшению и развитию проекта.

«Заключение» — содержит итоги о проделанной работе за прошедший семестр.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Тема проекта: «Разработка 3D-игры “Pacman-Three.js”».

Цель проекта: разработка 3D-игры с использованием технологий TypeScript, JavaScript и Three.js.

Основные задачи:

- разработка игры в сеттинге “Pacman” на TypeScript, JavaScript и Three.js;
- разработка игровых полей;
- изучение основ работы с библиотекой Three.js;
- разработка алгоритмов отрисовки и взаимодействия с элементами игры (игровое поле, пакман, призраки, еда);
- реализация алгоритмов логики движения и поведения основных элементов игры;
- миграция проекта прошлого семестра “Pacman-Forge” с проприетарной платформы Autodesk Forge на библиотеку со свободным исходным кодом Three.js;
- переработка проекта с нетипизированного функционального программирования (JavaScript) на типизированное объектно-ориентированное (TypeScript);
- улучшение визуальной составляющей игры: настройки камеры, сцены, освещения, разработка и загрузка моделей персонажей.

Этапы разработки:

1) изучение программного кода проекта прошлого семестра “Pacman-Forge”;
2) изучение библиотеки Three.js для разработки логики управления сценой и реализации отрисовки игры;
3) организация рабочего процесса между студентами для распределения и контролем за выполнением задач (github, trello, рабочие собрания);
4) изучение основ разработки на TypeScript;
5) реализация алгоритма отрисовки на сцену игровых полей и основных элементов игры;

6) разработка основных классов;
7) доработка и улучшение игровой логики и стабильности игры;
8) переработка алгоритма логики движения и поведения основных элементов игры;

ИНДИВИДУАЛЬНЫЕ ПЛАНЫ УЧАСТНИКОВ

Задачи по проекту были распределены следующим образом:

Хрусталев Г.Н. – менеджер проекта, ведущий разработчик, 3 курс:

- организация рабочего процесса между студентами;
- изучение основ работы с библиотекой Three.js;
- создание алгоритма загрузки моделей Blender в Three.js;
- интеграция TypeScript, классов, ООП и типизации в проект;
- разработка структуры программного кода;
- разработка основных классов игры;
- разработка рекурсивного алгоритма сборки блоков стен;
- разработка json базы данных для уровней;
- разработка обработчика для клавиш управления;
- реализация основных механик из оригинальной игры “Pacman”.

Зарубин А.Н. – дизайнер, 1 курс:

- изучение основ Three.js;
- изучение основ разработки на JavaScript;
- изучение основ работы с системой контроля версии git;
- разработка дизайна моделей пакмана и призраков;
- создание моделей пакмана и призраков Blinky, Pinky, Inky, Clyde в Blender
- разработка анимации поедания для пакмана;
- запись и монтирование демонстрационных видеороликов;
- создание документации.

Козырь С.К. – разработчик, 1 курс:

- изучение основ Three.js;
- изучение принципов работы с объектами и геометрией в Three.js
- изучение основ разработки на JavaScript и TypeScript;
- изучение основ работы с системой контроля версии git;
- изучение setInterval, функции animate, реализация игрового таймера;
- разработка обработчика для клавиш управления;

- создание обводки для стен;
- разработка алгоритма перемещения пакмана;
- создание документации.

Свинцицкий Р.Е. – дизайнер уровней, 1 курс

- изучение основ Three.js;
- изучение основ разработки на JavaScript;
- изучение основ работы с системой контроля версии git;
- разработка дизайна уровней;
- интеграция уровней и игровых объектов на сцену;
- разработка алгоритма для объединения блоков стен.

Селезнев В.К. – разработчик, 1 курс

- изучение основ Three.js;
- изучение основ разработки на JavaScript и TypeScript;
- изучение основ работы с системой контроля версии git;
- разработка основных методов игры;
- разработка модели вишни;
- проработка освещения сцены;
- создание документации.

ХОД РАБОТЫ

Создание основных элементов сцены

Разработка игры на Three.js начинается с подключения самой библиотеки и необходимых ее расширений, в нашем случае это управление камерой OrbitControls и загрузчик моделей GLTFLoader, для этого создадим файл который будет основным скриптом `main.js`.

Листинг 1.1

```
import * as THREE from './lib/three.module.js';
import { OrbitControls } from './lib/orbit-controls.three.module.js';
import { GLTFLoader } from './lib/gltf-loader.three.module.js';
```

После подключения необходимых библиотек, необходимо создать сцену, камеру, рендерер, подключить элементы управления камерой и сгенерировать блок viewer.

Листинг 1.2

```
const fov = 75;
const width = window.innerWidth * 0.8;
const height = window.innerHeight * 0.85;
const aspect = width / height;
const near = 0.1;
const far = 10000;

var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
var renderer = new THREE.WebGLRenderer();
renderer.setSize(width, height);
var controls = new OrbitControls(camera, renderer.domElement);
let viewerBox = document.getElementById('viewer');
viewerBox.appendChild(renderer.domElement);
camera.position.set(0, 0, 750);
controls.update();
```

В качестве простого освещения, отлично подойдет ненаправленный ambient light (общее освещение) белого цвета (“fafafa” в кодировке RGB) с 90% яркостью.

Листинг 1.3

```
let ambientLight = new THREE.AmbientLight(0xfafafa, 0.9);
scene.add(ambientLight);
```

После создания освещения на сцене можно добавлять на нее объекты – в нашем случае это будет куб, который играет роль заднего плана для 6-ти уровней. Меши (полигональные сетки) объектов в Three.js состоят из геометрии и материала. Очевидно, что для создания куба нам понадобится геометрия куба, а материал можно взять стандартный черного цвета (“000000” в палитре RGB).

Листинг 1.4

```
var geometry = new THREE.BoxGeometry(Params.CubeSize, Params.CubeSize,
Params.CubeSize);
var material = new THREE.MeshStandardMaterial({ color: 0x000000 });
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

Так как весь куб сплошного черного цвета – будет не лишним выделить ребра куба белым цветом, то есть создать его контуры.

Листинг 1.5

```
let edges = new THREE.EdgesGeometry(geometry);
let contour = new THREE.LineSegments(edges, new
THREE.LineBasicMaterial({ color:  0xfafafa }));
cube.add(contour);
```

Для того, чтобы картинка на экране менялась при ее изменениях на сцене, мы напишем несложную рекурсивную функцию, которая будет вызывать рендерер и генерировать следующий кадр.

Листинг 1.6

```
function animate() {
    requestAnimationFrame(animate);
    controls.update();
    renderer.render(scene, camera);
};
```

В результате можем запустить наш проект и увидеть в веб-браузере куб, который можно вращать, приближать и удалять при помощи мыши.

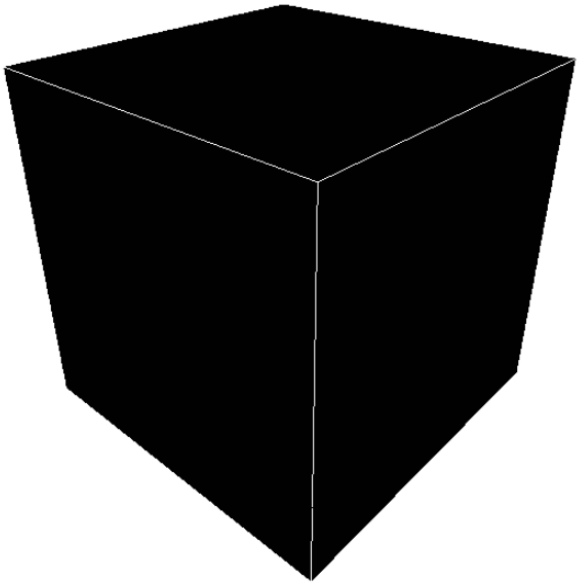

Рис. 1.1 — Добавление куба на сцену

Создание игровых уровней

Для создания игровых уровней необходимо сначала задать параметры куба: его размер, размер ячеек, параметры сетки и глубины (объема) объектов уровня. Для этого создадим скрипт на языке TypeScript `levels.ts`, который мы будем компилировать в JavaScript файл `levels.js`.

Листинг 2.1

```
export const Params = {
    CubeSize: 500,
    CellSize: 20,
    WallSize: 18,
    Rows: 25,
    Cols: 25,
    Depth: 20
}
export enum Objects {
    blank, // 0
```

Листинг 2.1 (продолжение)

```
    wall, // 1
    dot, // 2
    cherry, // 3
    powerup, // 4
    pacman, // 5
    blinky, // 6
    pinky, // 7
    inky, // 8
    clyde // 9
}
```

Так как карта уровня представляет собой сетку из блоков, достаточно удобно представить ее в качестве таблицы, поэтому разработка уровней осуществляется при помощи google таблиц с последующим копированием в массив json формата. Значения в клетках являются местами спавна определенных объектов: 1 – стена, 2 – единица еды, 3 – вишня, 4 – усиление пакмана (на данный момент не реализовано), 5 – пакман, 6 – призрак Blinky, 7 – призрак Pinky, 8 – призрак Inky, 9 – призрак Clyde.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
3	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	0
4	0	1	2	1	1	1	1	2	1	2	1	1	2	1	1	2	1	2	1	1	1	1	2	1	0
5	0	1	2	1	2	2	2	2	1	2	1	1	2	1	1	2	1	2	3	2	2	1	2	1	0
6	0	1	2	1	2	1	1	1	1	2	1	1	2	1	1	2	1	1	1	1	2	1	2	1	0
7	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	0
8	0	1	2	1	2	1	1	1	2	1	1	1	1	1	1	1	2	1	1	1	2	1	2	1	0
9	0	1	2	1	2	2	2	2	2	1	1	1	1	1	1	1	2	2	2	2	2	1	2	1	0
10	0	1	2	1	2	1	2	1	2	2	2	2	2	2	2	2	2	1	2	1	2	1	2	1	0
11	0	1	2	1	2	1	2	1	2	1	1	1	1	1	1	1	2	1	2	1	2	1	2	1	0
12	1	1	2	1	2	1	2	1	2	1	9	0	0	0	8	1	2	1	2	1	2	1	2	1	1
13	0	0	2	1	2	1	2	1	2	1	0	0	0	0	0	1	2	1	2	1	2	1	2	0	0
14	1	1	2	1	5	2	2	1	2	1	6	0	0	0	7	1	2	1	2	2	2	1	2	1	1
15	0	1	2	1	1	1	2	1	2	1	1	0	0	0	1	1	2	1	2	1	1	1	2	1	0
16	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	0
17	0	1	2	1	1	2	1	1	2	1	1	1	2	1	1	1	2	1	1	2	1	1	2	1	0
18	0	1	2	1	1	2	1	1	2	2	2	2	2	2	2	2	2	1	1	2	1	1	2	1	0
19	0	1	2	1	1	2	1	1	2	1	1	1	2	1	1	1	2	1	1	2	1	1	2	1	0
20	0	1	3	1	1	2	1	1	2	2	2	2	2	2	2	2	2	1	1	2	1	1	2	1	0
21	0	1	2	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	2	1	0
22	0	1	2	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	2	1	0
23	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	0
24	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
25	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

Рис. 2.1 — Передний уровень

Листинг 2.2

Массивы уровней

```
const FrontLevel = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 3, 2, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0],
    [1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 9, 0, 0, 0, 8, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1],
    [0, 0, 2, 1, 2, 1, 2, 1, 2, 1, 0, 0, 0, 0, 0, 1, 2, 1, 2, 1, 2, 1, 2, 0, 0],
    [1, 1, 2, 1, 5, 2, 2, 1, 2, 1, 6, 0, 0, 0, 7, 1, 2, 1, 2, 2, 2, 1, 2, 1, 1],
    [0, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 0, 0, 0, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 3, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];

const LeftLevel = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 3, 2, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 2, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 9, 0, 0, 0, 8, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 6, 0, 0, 0, 7, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
```

```
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];

const RightLevel = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 3, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 9, 0, 0, 0, 8, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 6, 0, 0, 0, 7, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 3, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];

const DownLevel = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
```

```
    [0, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 3, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 9, 0, 0, 0, 8, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 0, 0, 0, 1, 2, 1, 1, 3, 2, 2, 2, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 6, 0, 0, 0, 7, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];

const TopLevel = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 3, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 9, 0, 0, 0, 8, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 0, 0],
    [1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 6, 0, 0, 0, 7, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 1, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0],
```

Листинг 2.2 (продолжение)

```
    [0, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 3, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];

const BackLevel = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 3, 2, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0],
    [0, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 0],
    [1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 9, 0, 0, 0, 8, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1],
    [0, 0, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 0, 0],
    [1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 6, 0, 0, 0, 7, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1],
    [0, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 0],
    [0, 1, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0],
    [0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];
```

Функции по построению массива стен были позаимствованы из предыдущего проекта “Pacman-Forge”, поэтому они в этом отчете рассмотрены не будут.

Создание и добавление единиц еды на уровни

Для реализации единиц еды был разработан класс Dot, от которого наследуются классы Cherry и Powerup (Powerup и его игровая механика на данный

момент не реализованы). Мешем для Dot является обычная сфера белого цвета, в то время как для вишни была вручную создана собственная геометрия.

Листинг 3.1

```
type FoodType = 'dot' | 'cherry' | 'powerup' | 'ghost';

class Dot {
    public static readonly Size = 5;
    public readonly i: number;
    public readonly j: number;
    public mesh: THREE.Mesh;
    public readonly type: FoodType;

    constructor(i: number, j: number, type?: FoodType) {
        this.i = i;
        this.j = j;
        this.setMesh();
        this.type = type? type : 'dot';
    }
    public setMesh(): void {
        let sphere = new THREE.SphereGeometry(Dot.Size, Dot.Size, Dot.Size);
        let material  = new THREE.MeshStandardMaterial({ color: '#fafafa' } );
        this.mesh = new THREE.Mesh(sphere, material);
        this.mesh.position.set(this.getX(), this.getY(), 0);
    }
    public getX() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = this.j * Params.CellSize - (radius - delta);
        return x;
    }
    public getY() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let y = -this.i * Params.CellSize + (radius - delta);
        return y;
    }
}

class Cherry extends Dot {
```

Листинг 3.1 (продолжение)

```
    public static readonly Length = 5; // x
    public static readonly Height = 15; // y
```

```
    public static readonly Width = 15; // z

    constructor(i: number, j: number) {
        super(i, j, 'cherry');
        this.setMesh();
    }
    public setMesh() {
        let cherry_geometry = new THREE.SphereGeometry(Cherry.Length,
Cherry.Height, Cherry.Width);
        let cherry_material = new THREE.MeshStandardMaterial({ color: "#991c1c"
});
        let cherry1 = new THREE.Mesh(cherry_geometry, cherry_material);
        let cherry2 = new THREE.Mesh(cherry_geometry, cherry_material);
        cherry2.position.set(9,5,0); // ?

        let points1 = [];
        points1.push(new THREE.Vector3(0, Cherry.Length - 1, 0));
        points1.push(new THREE.Vector3(0, Cherry.Height - 1, 0));
        let pick_geometry1 = new THREE.BufferGeometry().setFromPoints(points1);
        let pick_material = new THREE.LineBasicMaterial({ color: '#35a12d',
linewidth: 5 });
        let pick1 = new THREE.Line(pick_geometry1, pick_material);
        cherry1.add(pick1);

        let curve = new THREE.EllipseCurve(0, 10, 8, 9.5, 3.5, -2.32 * Math.PI,
false, 1);
        let points2 = curve.getPoints(50);
        let pick_geometry2 = new THREE.BufferGeometry().setFromPoints(points2);
        let pick2 = new THREE.Line(pick_geometry2, pick_material);
        pick2.rotateY(Math.PI);
        pick2.rotateX(2 * Math.PI);
        cherry2.add(pick2);
        cherry1.add(cherry2);
        this.mesh = cherry1;
        this.mesh.position.set(this.getX(), this.getY(), 0);
    }
    public getX() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = this.j * Params.CellSize - (radius - delta) - Cherry.Length *
0.8;
        return x;
    }
```

Листинг 3.1 (продолжение)

```
    public getY() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let y = -this.i * Params.CellSize + (radius - delta) - Cherry.Height /
3;

        return y;
    }
}
```

Добавление единиц еды на уровни происходит при помощи прозрачных вспомогательных плоскостей через класс Game, который будет рассмотрен позже.

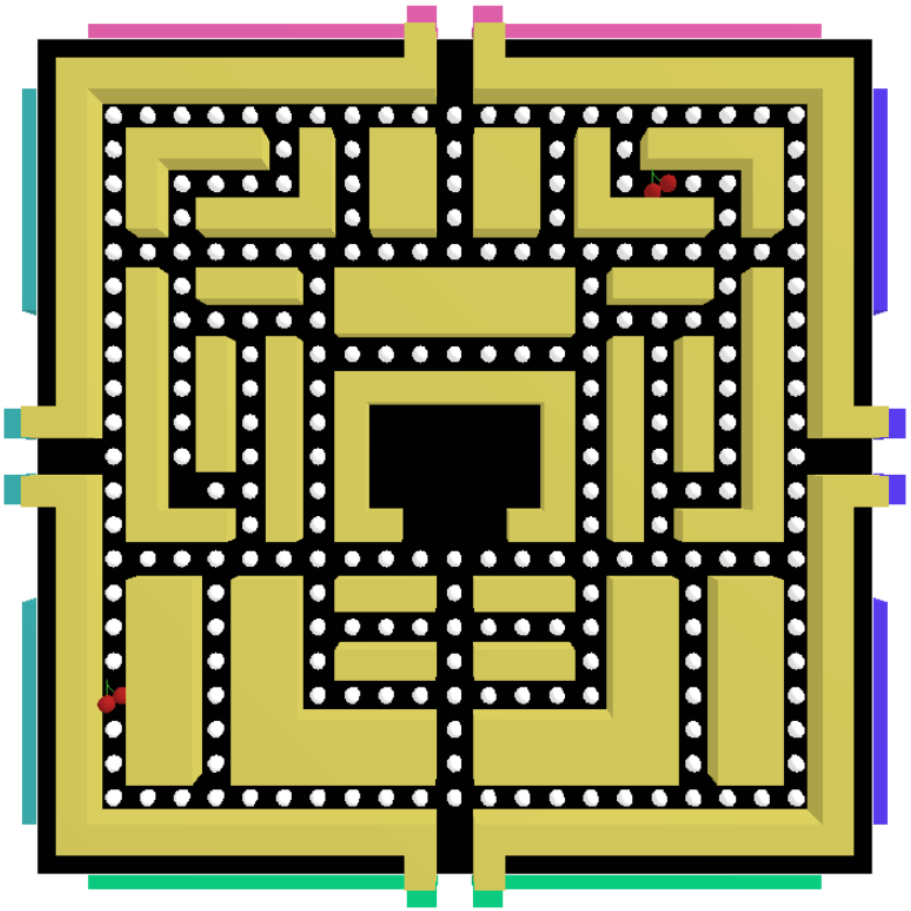

Рис. 3.1 — Добавление единиц еды и вишен на уровень

Создание абстрактного класса Entity

Для реализации персонажей с возможностью их передвижения по уровню и взаимодействием с ним, был разработан абстрактный класс Entity, от которого наследуются класс пакмана и класс призраков. Данный класс представлен в программном коде `entity.ts`.

Листинг 4.1

```
import * as THREE from './lib/three.module.js';
import { Params, Objects } from './levels.js';
import { Game } from './game.js';

export type Direction = 'up' | 'down' | 'left' | 'right' | 'none';

interface Cell {
    i: number,
    j: number
}

export abstract class Entity {
    public cell: Cell;
    protected moveDirection: Direction;
    public static Size: number;
    public type: Objects;
    private timer: any;
    protected model: THREE.Group;
    protected constructor(i?: number, j?: number, direction?: Direction) {
        this.cell = { i: (i? i : 0), j: (j? j : 0) };
        this.moveDirection = direction? direction : 'none';
    }

    public startMovement(direction: Direction) {
        if (!this.canMove(direction)) {
            return;
        } else {
            if (this.type == Objects.pacman)
                this.faceDirecton(direction);
            this.moveDirection = direction;
            clearInterval(this.timer);
            this.timer = null;
            this.timer = setInterval(() => {
                this.step();
                if (!this.canMove(direction))
```

Листинг 4.1 (продолжение)

```
                    clearInterval(this.timer);
                }, 200);
            }
        }

        private step() {
            switch(this.moveDirection) {
                case 'up':
                    this.cell.i -= 1;
                    break;
                case 'down':
                    this.cell.i += 1;
                    break;
                case 'left':
                    this.cell.j -= 1;
                    break;
                case 'right':
                    this.cell.j += 1;
                    break;
            }
               let point = this.getPointOnPlane(this.cell.i, this.cell.j); // TODO:
Заменить пересчет позиции на прирост координаты в сторону движения
            this.model.position.set(point.x, point.y, point.z);

            if (this.type == Objects.pacman)
                this.eatDot();
        }

        public canMove(direction: Direction): boolean {
            switch(direction) {
                case 'up':
                    if (this.cell.i == 0) {
                        return false;
                    } else {
                        return this.checkCell(this.cell.i - 1, this.cell.j);
                    }
                case 'down':
                    if (this.cell.i == Params.CubeSize / Params.CellSize - 1) {
                        return false;
                    } else {
                        return this.checkCell(this.cell.i + 1, this.cell.j);
                    }
```

```
            case 'left':
                if (this.cell.j == 0) {
                    return false;
                } else {
                    return this.checkCell(this.cell.i, this.cell.j - 1);
                }
            case 'right':
                if (this.cell.j == Params.CubeSize / Params.CellSize - 1) {
                    return false;
                } else {
                    return this.checkCell(this.cell.i, this.cell.j + 1);
                }
        }
    }

    protected checkCell(i: number, j: number): boolean {
        let level = Game.map[Game.curLevel].grid;
        if (level[i][j] == Objects.wall) {
            return false;
        } else {
            return true;
        }
    }

    private eatDot() {
        let predicate = dot => {
            return (dot.i == this.cell.i) && (dot.j == this.cell.j)
        };
        let dot = Game.levelDots[Game.curLevel].filter(predicate)[0];
        if (dot) {
            let index = Game.levelDots[Game.curLevel].indexOf(dot);
            Game.levelDots[Game.curLevel].splice(index, 1);
            dot.mesh.visible = false;
            Game.eat(dot.type);
        }
    }

    public abstract getX(j?: number);
    public abstract getY(i?: number);

    public faceDirecton(direction: Direction): void {
        let vector = this.calcRotation(direction);
```

Листинг 4.1 (продолжение)

```
        vector.x ? this.model.rotateX(vector.x) : {};
        vector.y ? this.model.rotateY(vector.y) : {};
        vector.z ? this.model.rotateY(vector.z) : {};
    }

    private calcRotation(direction: Direction) {
        let x = 0, y = 0, z = 0;
        switch(Game.curLevel)
        {
            case 'front':
                switch(direction) {
                    case 'up':
                        if (this.moveDirection == 'right')
                            x = Math.PI/2;
                        if (this.moveDirection == 'left')
                            x = -Math.PI/2;
                        if (this.moveDirection == 'down')
                            x = Math.PI;
                        break;
                    case 'down':
                        if (this.moveDirection == 'right')
                            x = -Math.PI/2;
                        if (this.moveDirection == 'left')
                            x = Math.PI/2;
                        if (this.moveDirection == 'up')
                            x = Math.PI;
                        break;
                    case 'left':
                        if (this.moveDirection == 'right')
                            x = Math.PI;
                        if (this.moveDirection == 'up')
                            x = Math.PI/2;
                        if (this.moveDirection == 'down')
                            x = -Math.PI/2;
                        break;
                    case 'right':
                        if (this.moveDirection == 'left')
                            x = Math.PI;
                        if (this.moveDirection == 'up')
                            x = -Math.PI/2;
                        if (this.moveDirection == 'down')
                            x = Math.PI/2;
```

```
                        break;
                }
                break;
            // TODO: Остальные грани
        }
        return new THREE.Vector3(x, y, z);
    }

    public rotateX(angle){
        this.model.rotateX(angle);
    }
    public rotateY(angle){
        this.model.rotateY(angle);
    }
    public rotateZ(angle){
        this.model.rotateZ(angle);
    }

    private getPointOnPlane(i: number, j: number) {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = j * Params.CellSize - (radius - delta);
        let y = -i * Params.CellSize + (radius - delta);
         let vector = new THREE.Vector3(x + Game.map[Game.curLevel].offset.x, y +
Game.map[Game.curLevel].offset.y,        Params.Depth        /        2        +
Game.map[Game.curLevel].offset.z);
              let  euler  =  new  THREE.Euler(Game.map[Game.curLevel].rotation.x,
Game.map[Game.curLevel].rotation.y, Game.map[Game.curLevel].rotation.z);
        vector.applyEuler(euler);
        return vector;
    }
}
```

Создание пакмана

Была разработана модель пакмана в программе Blender и экспортирована в игру в формате GLTF Binary (.glb) при помощи модуля GLTFLoader библиотеки Three.js.

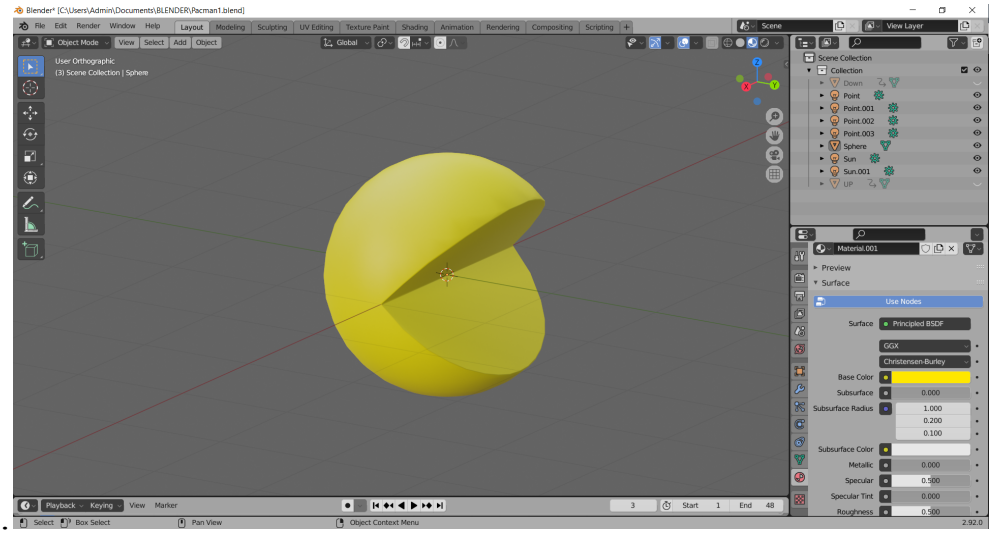

Рис. 5.1 — Модель пакмана в программе Blender

Для загрузки модели нам понадобится менеджер, который будет нам сообщать о процессе загрузки модели, а также сам GLTF Loader.

Листинг 5.1

Менеджер загрузки

```
const manager = new THREE.LoadingManager();
manager.onStart = function (url, itemsLoaded, itemsTotal) {
    console.log('Started loading file: ' + url + '.\nLoaded ' + itemsLoaded
+ ' of ' + itemsTotal + ' files.');
};
manager.onLoad = function () {
    console.log('Loading complete!');
};
manager.onProgress = function (url, itemsLoaded, itemsTotal) {
    console.log('Loading file: ' + url + '.\nLoaded ' + itemsLoaded + ' of
' + itemsTotal + ' files.');
};
manager.onError = function (url) {
    console.log('There was an error loading ' + url);
};
```

Листинг 5.2

Загрузка модели пакмана при помощи GLTF Loader

```
const loader = new GLTFLoader(manager);
loader.load('./models/Pacman.glb', function (gltf) {
    let pacman = gltf.scene;
    game.initPacman(pacman);
    cube.add(game.Pacman.getModel());
}, undefined, function (error) {
    console.error(error);
});
```

Далее мы создаем новый файл, в котором будет программный код класса пакмана – `pacman.ts`.

Листинг 5.3

```
import * as THREE from './lib/three.module.js';
import { Objects, Params } from './levels.js';
import { Direction, Entity } from './entity.js';

export class Pacman extends Entity {
    public spawnCell: { i: number, j: number };
    constructor(i?: number, j?: number, direction?: Direction) {
        super((i? i : 0), (j? j : 0), (direction? direction : 'none'));
        Pacman.Size = 10;
        this.type = Objects.pacman;
    }
    public override getX(j?: number) {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = (j? j : this.cell.j) * Params.CellSize - (radius - delta);
        return x;
    }
    public override getY(i?: number) {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let y = -(i? i : this.cell.i) * Params.CellSize + (radius - delta);
        return y;
    }
```

Листинг 5.3 (продолжение)

```
    public setModel(scene: THREE.Group) {
        this.model = scene;
    }
    public getModel() {
        return this.model;
    }
}
```

Создание призраков

Создание моделей призраков также велось в программе Blender. Модели призраков, как и сами призраки, отличаются друг от друга основным цветом: красный призрак – “Blinky”, розовый – “Pinky”, голубой – “Inky”, оранжевый – “Clyde”.

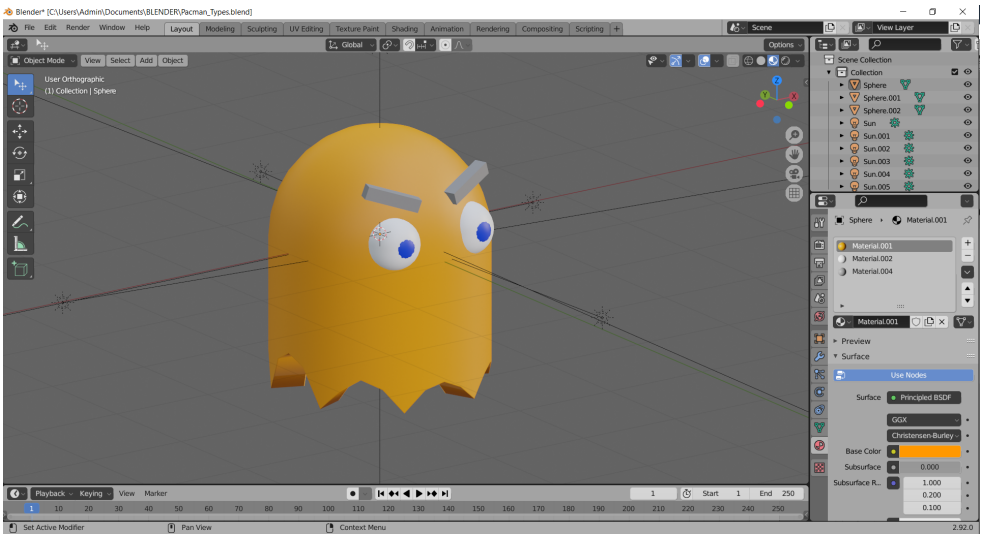

Рис. 6.1 — Модель призрака в “Clyde” программе Blender

Помимо основных моделей были также созданы модели убегающих призраков (на данный момент в игре это не реализовано). Загрузка моделей призраков осуществляется аналогичным пакману способом.

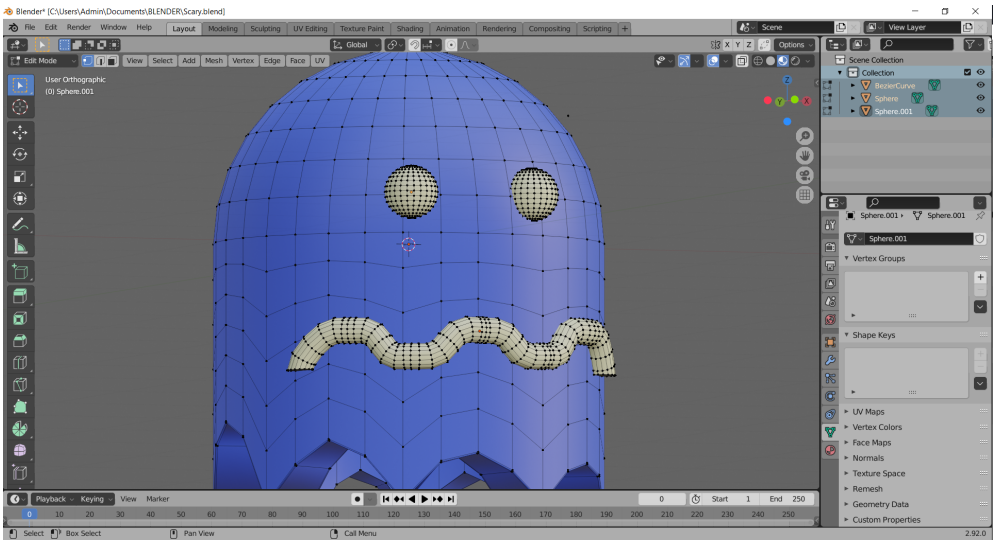

Рис. 6.2 — Топология модели убегающего призрака в программе Blender

Так как призраки между собой отличаются только моделью и игровым поведением – вполне логично создать наследующий Entity класс Ghost, и уже от него будут наследоваться и все остальные призраки. Создадим для этой задачи файл `ghost.ts` и напишем в нем следующий программный код.

Листинг 6.1

```
import * as THREE from './lib/three.module.js';
import { Entity, Direction } from './entity.js';
import { Objects, Params } from './levels.js';

export type GhostState = 'chase' | 'scatter' | 'fright';
export type GhostName = 'Blinky' | 'Pinky' | 'Inky' | 'Clyde';

export abstract class Ghost extends Entity {
    public state: GhostState;
    public spawnCell: { i: number, j: number };
    protected constructor(i?: number, j?: number, direction?: Direction) {
        super((i? i : 0), (j? j : 0), (direction? direction : 'none'));
        this.spawnCell = { i: (i? i : 0), j: (j? j : 0) };
        Ghost.Size = 8;
    }
    public override getX(j?: number) {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = (j? j : this.cell.j) * Params.CellSize - (radius - delta);
```

```
        return x;
    }
    public override getY(i?: number) {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let y = -(i? i : this.cell.i) * Params.CellSize + (radius - delta);
        return y;
    }
    public setModel(scene: THREE.Group) {
        this.model = scene;
    }
    public getModel() {
        return this.model;
    }
}
export class Blinky extends Ghost {
    constructor(i?: number, j?: number, direction?: Direction) {
        super((i? i : 0), (j? j : 0), (direction? direction : 'none'));
        this.type = Objects.blinky;
    }
}
export class Pinky extends Ghost {
    constructor(i?: number, j?: number, direction?: Direction) {
        super((i? i : 0), (j? j : 0), (direction? direction : 'none'));
        this.type = Objects.pinky;
    }
}
export class Inky extends Ghost {
    constructor(i?: number, j?: number, direction?: Direction) {
        super((i? i : 0), (j? j : 0), (direction? direction : 'none'));
        this.type = Objects.inky;
    }
}
export class Clyde extends Ghost {
    constructor(i?: number, j?: number, direction?: Direction) {
        super((i? i : 0), (j? j : 0), (direction? direction : 'none'));
        this.type = Objects.clyde;
    }
}
```

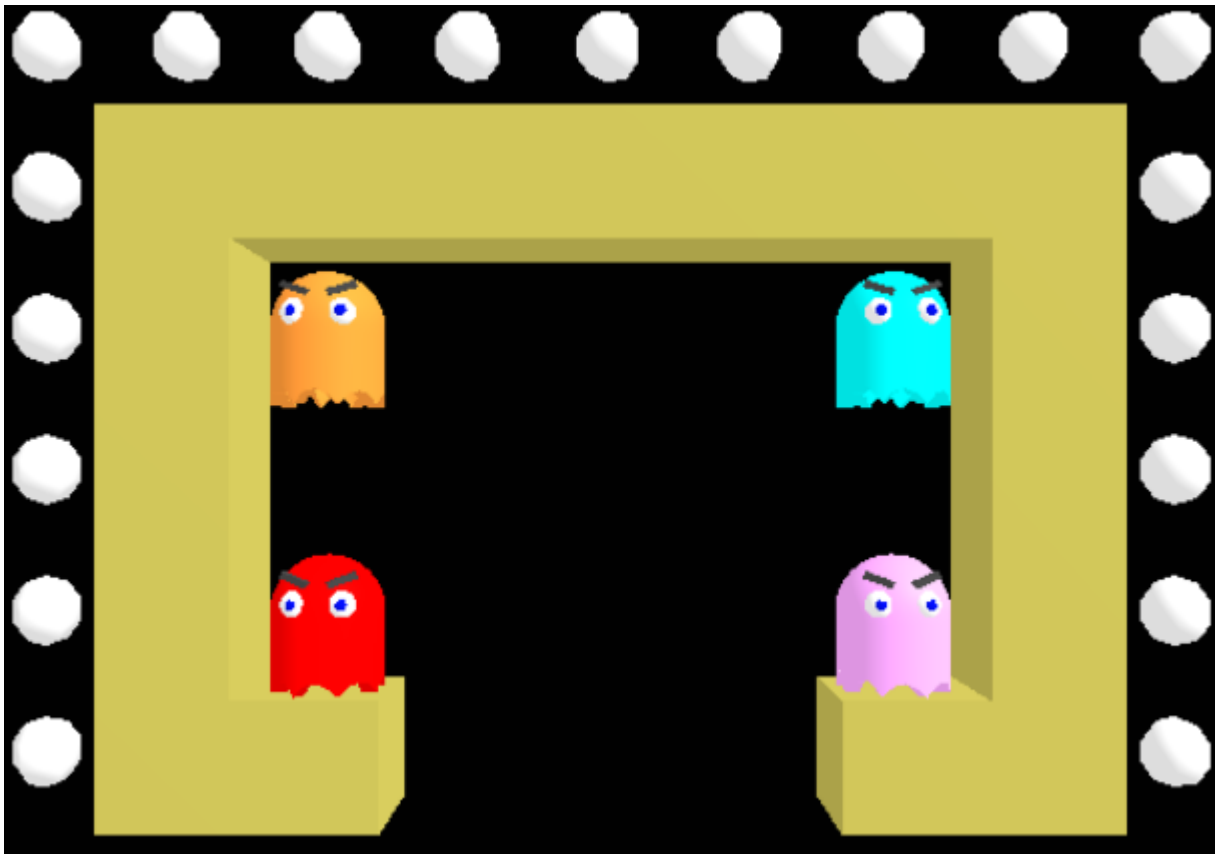

Рис. 6.3 — Загруженные на уровень модели призраков

Создание класса игры

Класс Game представляет собой основной связующий управляющий класс, который отвечает за обработку игровой логики и поведения. Ниже представлен программный код файла с разработанным классом игры `game.ts`.

Листинг 7.1

```
import * as THREE from './lib/three.module.js';
import { Objects, Params, Level, Map, MAP, LevelType  } from './levels.js';
import { Pacman } from './pacman.js';
import { Ghost, Blinky, Pinky, Inky, Clyde, GhostName } from './ghosts.js';


interface Point {
    x: number;
    y: number;
}

interface Index {
    i: number;
```

```
    j: number;
}

interface LevelDots {
    'front': Dot[],
    'back': Dot[],
    'right': Dot[],
    'left': Dot[],
    'top': Dot[],
    'bottom': Dot[]
}

type FoodType = 'dot' | 'cherry' | 'powerup' | 'ghost';

class Dot {
    public static readonly Size = 5;
    public readonly i: number;
    public readonly j: number;
    public mesh: THREE.Mesh;
    public readonly type: FoodType;

    constructor(i: number, j: number, type?: FoodType) {
        this.i = i;
        this.j = j;
        this.setMesh();
        this.type = type? type : 'dot';
    }
    public setMesh(): void {
        let sphere = new THREE.SphereGeometry(Dot.Size, Dot.Size, Dot.Size);
        let material  = new THREE.MeshStandardMaterial({ color: '#fafafa' } );
        this.mesh = new THREE.Mesh(sphere, material);
        this.mesh.position.set(this.getX(), this.getY(), 0);
    }
    public getX() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = this.j * Params.CellSize - (radius - delta);
        return x;
    }
    public getY() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
```

```
        let y = -this.i * Params.CellSize + (radius - delta);
        return y;
    }
}

class Cherry extends Dot {
    public static readonly Length = 5; // x
    public static readonly Height = 15; // y
    public static readonly Width = 15; // z

    constructor(i: number, j: number) {
        super(i, j, 'cherry');
        this.setMesh();
    }
    public setMesh() {
                let cherry_geometry = new THREE.SphereGeometry(Cherry.Length,
Cherry.Height, Cherry.Width);
         let cherry_material = new THREE.MeshStandardMaterial({ color: "#991c1c"
});
        let cherry1 = new THREE.Mesh(cherry_geometry, cherry_material);
        let cherry2 = new THREE.Mesh(cherry_geometry, cherry_material);
        cherry2.position.set(9,5,0); // ?

        let points1 = [];
        points1.push(new THREE.Vector3(0, Cherry.Length - 1, 0));
        points1.push(new THREE.Vector3(0, Cherry.Height - 1, 0));
        let pick_geometry1 = new THREE.BufferGeometry().setFromPoints(points1);
            let pick_material = new THREE.LineBasicMaterial({ color: '#35a12d',
linewidth: 5 });
        let pick1 = new THREE.Line(pick_geometry1, pick_material);
        cherry1.add(pick1);

         let curve = new THREE.EllipseCurve(0, 10, 8, 9.5, 3.5, -2.32 * Math.PI,
false, 1);
        let points2 = curve.getPoints(50);
        let pick_geometry2 = new THREE.BufferGeometry().setFromPoints(points2);
        let pick2 = new THREE.Line(pick_geometry2, pick_material);
        pick2.rotateY(Math.PI);
        pick2.rotateX(2 * Math.PI);
        cherry2.add(pick2);
        cherry1.add(cherry2);
        this.mesh = cherry1;
```

Листинг 7.1 (продолжение)

```
        this.mesh.position.set(this.getX(), this.getY(), 0);
    }
    public getX() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
          let x = this.j * Params.CellSize - (radius - delta) - Cherry.Length *
0.8;
        return x;
    }
    public getY() {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
         let y = -this.i * Params.CellSize + (radius - delta) - Cherry.Height /
3;
        return y;
    }
}

export class Game {
    public static readonly map: Map = MAP;
    public static levelDots: LevelDots;
    public static curLevel: LevelType;
    public static score: number;
    public static scoreText: HTMLElement;
    private levels: Level[];
    private Pacman: Pacman;
    public Blinky: Blinky;
    public Pinky: Pinky;
    public Inky: Inky;
    public Clyde: Clyde;
    private sides = [ 'front', 'back', 'right', 'left', 'top', 'bottom' ];
    private static planes = {};

    constructor() {
        Game.curLevel = 'front';
        Game.score = 0;
        Game.scoreText = document.getElementById('score');
        this.setLevels();
    }

    public startGame() {
        // TODO
```

```
    }

    public restartGame() {
        // TODO
    }

    public findObjects(object: Objects, grid: number[][]) {
        let array: Index[] = [];
        for (let i = 0; i < Params.CubeSize/Params.CellSize; i++)
            for (let j = 0; j < Params.CubeSize/Params.CellSize; j++)
                if (grid[i][j] == object)
                    array.push({ i: i, j: j });
        return array;
    }

    private loadDots(): void {
        let dots: Dot[] = [];
        this.clearLevelDots();
        for (let level of this.levels)
        {
                let dotsIndexes = this.findObjects(Objects.dot, level.grid); //
Поиск всех единиц еды
            for (let index of dotsIndexes)
            {
                let dot = new Dot(index.i, index.j);
                dots.push(dot);
            }
             let cherriesIndexes = this.findObjects(Objects.cherry, level.grid);
// Поиск всех вишен
            for (let index of cherriesIndexes)
            {
                let cherry = new Cherry(index.i, index.j);
                dots.push(cherry);
            }
            Game.levelDots[level.name] = dots;
            dots = [];
        }
    }

    public static eat(type: FoodType) {
        switch(type) {
            case 'dot':
```

```
                Game.score += 10;
                Game.scoreText.innerText = `Счет: ${Game.score}`;
                break;
            case 'cherry':
                Game.score += 100;
                Game.scoreText.innerText = `Счет: ${Game.score}`;
                break;
            case 'powerup':
                // TODO
                break;
            case 'ghost':
                // TODO
                break;
        }
    }

    public drawLevelPlanes() {
                    let geometry = new THREE.PlaneGeometry(Params.CubeSize,
Params.CubeSize);
                let material = new THREE.MeshStandardMaterial({color: 0xffffff,
transparent: true, opacity: 0.0 });

        for (let side of this.sides)
        {
                let offset = { // Добавление дополнительного смещения в половину
высоты стены
                    x: Game.map[side].offset.x ? (Game.map[side].offset.x > 0 ?
Game.map[side].offset.x + Params.Depth/2 : Game.map[side].offset.x -
Params.Depth/2) : 0,
                    y: Game.map[side].offset.y ? (Game.map[side].offset.y > 0 ?
Game.map[side].offset.y + Params.Depth/2 : Game.map[side].offset.y -
Params.Depth/2) : 0,
                    z: Game.map[side].offset.z ? (Game.map[side].offset.z > 0 ?
Game.map[side].offset.z + Params.Depth/2 : Game.map[side].offset.z -
Params.Depth/2) : 0
                }
                Game.planes[side] = new THREE.Mesh(geometry, material);
                Game.planes[side].position.set(offset.x, offset.y, offset.z);
                                        Game.planes[side].setRotationFromEuler(new
THREE.Euler(Game.map[side].rotation.x,                 Game.map[side].rotation.y,
Game.map[side].rotation.z));
        }
```

```
        this.loadDots();

        this.sides.forEach(side => {
            for (let dot of Game.levelDots[side])
                Game.planes[side].add(dot.mesh);
        });

        let planesArray = [];
        this.sides.forEach(side => {
            planesArray.push(Game.planes[side]);
        });
        return planesArray;
    }

    private drawWalls() {
        let levelWalls = [];
        for (let level of this.levels)
        {
                let walls = this.findObjects(Objects.wall, level.grid); // Поиск
всех блоков стен
            let checkedCells = []; this.clearCheckedCells(checkedCells);
            let wallMeshes = [];
            let tempWall: Point[] = [];
            for (let block of walls) // Обход по каждому блоку стены
            {
                let i = block.i; let j = block.j;
                if (level.grid[i][j] != checkedCells[i][j]) {
                    checkedCells[i][j] = Objects.wall;
                    this.tempWallAdd(tempWall, i, j);

                                    this.follow(level.grid, 'left', i, j, tempWall,
checkedCells);
                                this.follow(level.grid, 'right', i, j, tempWall,
checkedCells)
                    let wall = this.truncateWall(tempWall);
                    tempWall = [];
                    let shape = this.drawPath(wall);
                    let extrudeSettings = {
                        steps: 1,
                        depth: Params.Depth,
                        bevelEnabled: false,
                    };
```

Листинг 7.1 (продолжение)

```
                                let geometry = new THREE.ExtrudeGeometry(shape,
extrudeSettings);
                            let material = new THREE.MeshStandardMaterial({ color:
level.color });
                        let wallMesh = new THREE.Mesh(geometry, material);
                                wallMesh.position.set(level.offset.x, level.offset.y,
level.offset.z);
                                                    wallMesh.rotation.setFromVector3(new
THREE.Vector3(level.rotation.x, level.rotation.y, level.rotation.z));
                        wallMeshes.push(wallMesh);
                    }
                }
                levelWalls.push(wallMeshes);
            }
            return levelWalls;
        }

        public drawLevelWalls() {
            let wallArray = [];
            let levelWalls = this.drawWalls();
            for (let level of levelWalls)
            {
                for (let walls of level)
                {
                    wallArray.push(walls);
                }
            }
            return wallArray;
        }

        public initPacman(scene): void {
                                let position = this.findObjects(Objects.pacman,
Game.map[Game.curLevel].grid)[0];
            this.Pacman = new Pacman(position.i, position.j, 'right');
                            this.Pacman.spawnCell = this.findObjects(Objects.pacman,
Game.map[Game.curLevel].grid)[0];
            let pacman = scene;
            pacman.scale.set(Pacman.Size, Pacman.Size, Pacman.Size);
            let point = this.getPointOnPlane(position.i, position.j, Game.curLevel);
            pacman.position.set(point.x, point.y, point.z);
            this.Pacman.setModel(pacman);
            pacman.rotateY(-Math.PI / 2);
        }
```

Листинг 7.1 (продолжение)

```
    public initGhost(scene, ghost: GhostName): void {
        let position: Index;
        let point: THREE.Vector3;
        switch(ghost) {
            case 'Blinky':
                                    position  =  this.findObjects(Objects.blinky,
Game.map[Game.curLevel].grid)[0];
                this.Blinky = new Blinky(position.i, position.j);
                let blinky = scene;
                blinky.scale.set(Ghost.Size, Ghost.Size, Ghost.Size);
                            point  =  this.getPointOnPlane(position.i,  position.j,
Game.curLevel);
                blinky.position.set(point.x, point.y, point.z);
                this.Blinky.setModel(blinky);
                break;
            case 'Pinky':
                                    position  =  this.findObjects(Objects.pinky,
Game.map[Game.curLevel].grid)[0];
                this.Pinky = new Pinky(position.i, position.j);
                let pinky = scene;
                pinky.scale.set(Ghost.Size, Ghost.Size, Ghost.Size);
                            point  =  this.getPointOnPlane(position.i,  position.j,
Game.curLevel);
                pinky.position.set(point.x, point.y, point.z);
                this.Pinky.setModel(pinky);
                break;
            case 'Inky':
                                    position  =  this.findObjects(Objects.inky,
Game.map[Game.curLevel].grid)[0];
                this.Inky = new Inky(position.i, position.j);
                let inky = scene;
                inky.scale.set(Ghost.Size, Ghost.Size, Ghost.Size);
                            point  =  this.getPointOnPlane(position.i,  position.j,
Game.curLevel);
                inky.position.set(point.x, point.y, point.z);
                this.Inky.setModel(inky);
                break;
            case 'Clyde':
                                    position  =  this.findObjects(Objects.clyde,
Game.map[Game.curLevel].grid)[0];
                this.Clyde = new Clyde(position.i, position.j);
                let clyde = scene;
```

Листинг 7.1 (продолжение)

```
                clyde.scale.set(Ghost.Size, Ghost.Size, Ghost.Size);
                        point = this.getPointOnPlane(position.i, position.j,
Game.curLevel);
                clyde.position.set(point.x, point.y, point.z);
                this.Clyde.setModel(clyde);
                break;
        }
    }

    private clearCheckedCells(checkedCells): void {
        for (let i = 0; i < Params.Rows; i++)
            checkedCells[i] = [];
    }

    private clearLevelDots(): void {
        Game.levelDots = {
            'front': [],
            'back': [],
            'right': [],
            'left': [],
            'top': [],
            'bottom': []
        }
    }

    private tempWallAdd(tempWall: Point[], i: number, j: number): void {
        let cellDelta = Params.CellSize / 2;
        let wallDelta = Params.WallSize / 2;
        let radius = Params.CubeSize / 2;
          let center = { x: j * Params.CellSize - (radius - cellDelta), y: -i *
Params.CellSize + (radius - cellDelta)}
         tempWall.push({ x: center.x - wallDelta, y: center.y + wallDelta }); //
left top
         tempWall.push({ x: center.x + wallDelta, y: center.y + wallDelta }); //
right top
         tempWall.push({ x: center.x + wallDelta, y: center.y - wallDelta }); //
right bottom
         tempWall.push({ x: center.x - wallDelta, y: center.y - wallDelta }); //
left bottom
    }
```

Листинг 7.1 (продолжение)

```
    private follow(level, direction, i, j, tempWall, checkedCells): void { //
Рекурсивный метод по сборке стены
        if (level[i][j] == Objects.wall) {
            switch (direction) {
                case 'right':
                    if (j + 1 < Params.CubeSize / Params.CellSize)
                    {
                            if (level[i][j + 1] == Objects.wall && level[i][j + 1]
!= checkedCells[i][j + 1]) { // Если справа стена
                             this.tempWallAdd(tempWall, i, j + 1);
                             checkedCells[i][j + 1] = Objects.wall;
                                  this.follow(level, 'right', i, j + 1, tempWall,
checkedCells);
                        } else {
                                      this.follow(level, 'down', i, j, tempWall,
checkedCells);
                        }
                    }
                    break;
                case 'left':
                    if (j - 1 >= 0)
                    {
                            if (level[i][j - 1] == Objects.wall && level[i][j - 1]
!= checkedCells[i][j - 1]) { // Если слева стена
                             this.tempWallAdd(tempWall, i, j - 1);
                             checkedCells[i][j - 1] = Objects.wall;
                                   this.follow(level, 'left', i, j - 1, tempWall,
checkedCells);
                        } else {
                                      this.follow(level, 'down', i, j, tempWall,
checkedCells);
                        }
                    }
                    break;
                case 'down':
                    if (i + 1 < Params.CubeSize / Params.CellSize)
                    {
                            if (level[i + 1][j] == Objects.wall && level[i + 1][j]
!= checkedCells[i + 1][j]) { // Если снизу стена
                             this.tempWallAdd(tempWall, i + 1, j);
                             checkedCells[i + 1][j] = Objects.wall;
```

Листинг 7.1 (продолжение)

```
                                this.follow(level, 'down', i + 1, j, tempWall,
checkedCells);
                                this.follow(level, 'left', i + 1, j, tempWall,
checkedCells);
                               this.follow(level, 'right', i + 1, j, tempWall,
checkedCells);
                        }
                    }
                    break;
                case 'up': // TODO: FIX
                    if (i - 1 >= 0)
                    {
                           if (level[i - 1][j] == Objects.wall && level[i - 1][j]
!= checkedCells[i - 1][j]) { // Если сверху стена
                            this.tempWallAdd(tempWall, i - 1, j);
                            checkedCells[i - 1][j] = Objects.wall;
                                this.follow(level, 'up', i - 1, j, tempWall,
checkedCells);
                               this.follow(level, 'right', i - 1, j, tempWall,
checkedCells);
                                this.follow(level, 'left', i - 1, j, tempWall,
checkedCells);
                        }
                    }
                break;
            }
        }
    }

     private findObjectsAround(level: number[][], i: number, j: number, object:
Objects) {
        let result: string[] = [];
        // left
        if (j - 1 >= 0)
            if (level[i][j - 1] == object)
                result.push('left');
        // right
        if (j + 1 < Params.CubeSize / Params.CellSize)
            if (level[i][j + 1] == object)
                result.push('right');
        // up
        if (i - 1 >= 0)
```

```
            if (level[i - 1][j] == object)
                result.push('up');
        // down
        if (i + 1 < Params.CubeSize / Params.CellSize)
            if (level[i + 1][j] == object)
                result.push('down');
        return result;
    }

    private truncateWall(tempWall) {
        let wall = [];
        for (let i = 0; i < tempWall.length; i++) {
            let count = 0;
            let point = tempWall[i];
            let gap = Params.CellSize - Params.WallSize;

                let top = this.checkPointToSkip(tempWall, point.x, point.y +
Params.WallSize) || this.checkPointToSkip(tempWall, point.x, point.y + gap);
                let bottom = this.checkPointToSkip(tempWall, point.x, point.y -
Params.WallSize) || this.checkPointToSkip(tempWall, point.x, point.y - gap);
                    let right = this.checkPointToSkip(tempWall, point.x +
Params.WallSize, point.y) || this.checkPointToSkip(tempWall, point.x + gap,
point.y);
                    let left = this.checkPointToSkip(tempWall, point.x -
Params.WallSize, point.y) || this.checkPointToSkip(tempWall, point.x - gap,
point.y);
            let corners = this.checkPointToSkip(tempWall, point.x - gap, point.y
+ gap) ||
                        this.checkPointToSkip(tempWall, point.x + gap, point.y
+ gap) ||
                        this.checkPointToSkip(tempWall, point.x + gap, point.y
- gap) ||
                        this.checkPointToSkip(tempWall, point.x - gap, point.y
- gap);

            if (top) count++;
            if (bottom) count++;
            if (right) count++;
            if (left) count++;
            if (corners) count++;

            if (count < 5) wall.push(tempWall[i]);
```

```
        }
        return wall;
    }

    private checkPointToSkip(tempWall: Point[], x: number, y: number) { // Пропуск точек внутри фигуры
        return tempWall.some(item => item.x == x && item.y == y);
    }

    private drawPath(wall) {
        let head = wall[0];
        let shape = new THREE.Shape();
        shape.moveTo(head.x, head.y);
        this.checkPath(head, wall, shape);
        return shape;
    }

    private checkPath(head, wall, shape) {
            let topC = this.findPointAround(head.x, head.y + (Params.CellSize - Params.WallSize), wall);
        let topW = this.findPointAround(head.x, head.y + Params.WallSize, wall);
                let rightC = this.findPointAround(head.x + (Params.CellSize - Params.WallSize), head.y, wall);
            let rightW = this.findPointAround(head.x + Params.WallSize, head.y, wall);
                let leftC = this.findPointAround(head.x - (Params.CellSize - Params.WallSize), head.y, wall);
            let leftW = this.findPointAround(head.x - Params.WallSize, head.y, wall);
           let downC = this.findPointAround(head.x, head.y - (Params.CellSize - Params.WallSize), wall);
            let downW = this.findPointAround(head.x, head.y - Params.WallSize, wall);

          let pointsAround = [rightC, rightW, downC, downW, leftC, leftW, topC, topW];
        for (let item of pointsAround) {
            if (!!item) { // Проверка на undefined
                wall.splice(wall.indexOf(item), 1);
                shape.lineTo(item.x, item.y);
                this.checkPath(item, wall, shape);
                break;
            }
        }
```

```
    }

    private findPointAround(x, y, wall) {
        return wall.find(item => item.x == x && item.y == y);
    }

    private getPointOnPlane(i: number, j: number, level: LevelType) {
        let delta = Params.CellSize / 2;
        let radius = Params.CubeSize / 2;
        let x = j * Params.CellSize - (radius - delta);
        let y = -i * Params.CellSize + (radius - delta);
            let vector = new THREE.Vector3(x + Game.map[level].offset.x, y +
Game.map[level].offset.y, Params.Depth / 2 + Game.map[level].offset.z);
                let euler = new THREE.Euler(Game.map[level].rotation.x,
Game.map[level].rotation.y, Game.map[level].rotation.z);
        vector.applyEuler(euler);
        return vector;
    }

    public static removeDot(dot: Dot, level: LevelType) {
            let object = Game.planes[level].children.find(item => item.mesh ==
dot.mesh);
        console.log(object);
    }

    // Get и Set методы
    public static getLevel() {
        return Game.curLevel;
    }
    public static setLevel(level: LevelType) {
        Game.curLevel = level;
    }
    /*public setLevel(level: LevelType) {
        this.switchLevel(); // TODO: Переход на другой уровень
    }*/
    public getLevels() {
        return this.levels;
    }
    private setLevels() {
        this.levels = [];
        this.levels.push(Game.map['front']);
        this.levels.push(Game.map['back']);
```

Листинг 7.1 (продолжение)

```
        this.levels.push(Game.map['right']);
        this.levels.push(Game.map['left']);
        this.levels.push(Game.map['top']);
        this.levels.push(Game.map['bottom']);
    }
}
```

РЕЗУЛЬТАТЫ

В результате проделанной работы мы создали игру Pacman-Three.js. Главная идея проекта заключается в том, чтобы реализовать куб, на каждой грани которого размещена игровая зона в виде несложного лабиринта, таким образом, чтобы объединить все уровни между собой вы одной большое поле. Игра сохраняет оригинальную стилистику классической игры “Pac-Man”. Задача игры все еще заключается в том, чтобы убегать от призраков, собирать единицы еды и получать за это очки.

Разработанная игра позволила нам углубить свои знания в создании игр, программировании на TypeScript и разработке ПО с помощью библиотеки Three.js.

Дальнейшие планы по разработке проекта включают в себя следующие задачи:

- доработка алгоритмов поведения призраков и их интеграция в игру;
- улучшение системы перемещений моделей, добавление плавности для анимаций;
- реализация механизма перехода на другой уровень;
- разработка основных игровых составляющих: кнопка начала игры, система жизней пакмана, перезапуск игры, класс powerup и система поедания призраков;
- разработка системы рекордов и таблицы рекордсменов;
- доработка интерфейса, адаптивность сайта, создание системы управления для мобильных устройств.

Ссылки на проект:

1. Trello: https://trello.com/b/S3EYgUL8/проектная-деятельность
2. GitHub репозиторий: https://github.com/JoshTD/PA-Pacman-ThreeJS
3. Хостинг игры: http://pacman.mpu-cloud.ru

ЗАКЛЮЧЕНИЕ

Команда 3-его и 1-го курса «Центра САПР-разработки» выполнила работу по разработке 3D-игры “Pacman-Three.js”. В процессе разработки нами были изучены принципы работы с библиотекой Three.js, основы применения объектно-ориентированного программирования на TypeScript для разработки 3D-игры, получены особо важные навыки работы в команде и совместной разработке программного кода.