class Game {

constructor() {

this.elementDatabase = [

{symbol: 'H', name: 'Hydrogen', latin: 'Hydrogenium', number: 1, valency: 1},

{symbol: 'He', name: 'Helium', latin: 'Helium', number: 2, valency: 0},

{symbol: 'Li', name: 'Lithium', latin: 'Lithium', number: 3, valency: 1},

{symbol: 'Be', name: 'Beryllium', latin: 'Beryllium', number: 4, valency: 2},

{symbol: 'B', name: 'Boron', latin: 'Borum', number: 5, valency: 3},

{symbol: 'C', name: 'Carbon', latin: 'Carboneum', number: 6, valency: 4},

{symbol: 'N', name: 'Nitrogen', latin: 'Nitrogenium', number: 7, valency: 3},

{symbol: 'O', name: 'Oxygen', latin: 'Oxygenium', number: 8, valency: 2},

{symbol: 'F', name: 'Fluorine', latin: 'Fluorum', number: 9, valency: 1},

{symbol: 'Ne', name: 'Neon', latin: 'Neon', number: 10, valency: 0},

{symbol: 'Na', name: 'Sodium', latin: 'Natrium', number: 11, valency: 1},

{symbol: 'Mg', name: 'Magnesium', latin: 'Magnesium', number: 12, valency: 2},

{symbol: 'Al', name: 'Aluminum', latin: 'Aluminium', number: 13, valency: 3},

{symbol: 'Si', name: 'Silicon', latin: 'Silicium', number: 14, valency: 4},

{symbol: 'P', name: 'Phosphorus', latin: 'Phosphorus', number: 15, valency: 3},

{symbol: 'S', name: 'Sulfur', latin: 'Sulfur', number: 16, valency: 2},

{symbol: 'Cl', name: 'Chlorine', latin: 'Chlorum', number: 17, valency: 1},

{symbol: 'Ar', name: 'Argon', latin: 'Argonium', number: 18, valency: 0},

{symbol: 'K', name: 'Potassium', latin: 'Kalium', number: 19, valency: 1},

{symbol: 'Ca', name: 'Calcium', latin: 'Calcium', number: 20, valency: 2},

{symbol: 'Sc', name: 'Scandium', latin: 'Scandium', number: 21, valency: 3},

{symbol: 'Ti', name: 'Titanium', latin: 'Titanium', number: 22, valency: 4},

{symbol: 'V', name: 'Vanadium', latin: 'Vanadium', number: 23, valency: 5},

{symbol: 'Cr', name: 'Chromium', latin: 'Chromium', number: 24, valency: 3}

];

this.tiles = [];

this.selectedTiles = [];

this.score = 0;

this.timeLeft = 60;

this.level = 1;

this.gameOver = false;

this.lastTime = 0;

this.combo = 0;

this.lastMatchTime = 0;

this.particles = [];

this.scorePopups = [];

this.gameMode = 'symbol-name';

this.hintUsed = false;

}

setup() {

this.createBackground();

this.createUI();

this.createGrid();

}

createBackground() {

const bg = createTilingSprite('background\_tile', app.screen.width, app.screen.height);

app.stage.addChild(bg);

}

createUI() {

const scoreIcon = createSprite('score\_icon', {x: 50, y: 30, width: 40});

app.stage.addChild(scoreIcon);

this.scoreText = createText('Score: 0', {

fontSize: 24,

fill: 0xFFFFFF,

fontWeight: 'bold'

}, {x: 100, y: 35});

app.stage.addChild(this.scoreText);

this.comboText = createText('Combo: x1', {

fontSize: 20,

fill: 0xFFD700,

fontWeight: 'bold'

}, {x: 50, y: 65});

app.stage.addChild(this.comboText);

this.levelText = createText('Level: 1', {

fontSize: 20,

fill: 0x00FF00,

fontWeight: 'bold'

}, {x: app.screen.width / 2 - 40, y: 35});

app.stage.addChild(this.levelText);

this.modeText = createText('Mode: Symbol ↔ Name', {

fontSize: 16,

fill: 0x00FFFF,

fontWeight: 'bold'

}, {x: app.screen.width / 2 - 60, y: 60});

app.stage.addChild(this.modeText);

const timerIcon = createSprite('timer\_icon', {x: app.screen.width - 150, y: 30, width: 40});

app.stage.addChild(timerIcon);

this.timerText = createText('Time: 60', {

fontSize: 24,

fill: 0xFFFFFF,

fontWeight: 'bold'

}, {x: app.screen.width - 100, y: 35});

app.stage.addChild(this.timerText);

if (this.timeLeft <= 10) {

this.timerText.style.fill = 0xFF0000;

}

}

createGrid() {

const tileWidth = 120;

const tileHeight = 90;

const gridSize = Math.min(4 + Math.floor(this.level / 3), 6);

const spacing = 10;

const startX = (app.screen.width - (gridSize \* (tileWidth + spacing))) / 2;

const startY = 120;

let tileData = [];

const elementsToUse = this.elementDatabase.slice(0, Math.min(8 + this.level, 24));

elementsToUse.forEach(element => {

if (this.gameMode === 'symbol-name') {

tileData.push({type: 'symbol', value: element.symbol, pair: element.name, element: element});

tileData.push({type: 'name', value: element.name, pair: element.symbol, element: element});

} else if (this.gameMode === 'number-symbol') {

tileData.push({type: 'number', value: element.number.toString(), pair: element.symbol, element: element});

tileData.push({type: 'symbol', value: element.symbol, pair: element.number.toString(), element: element});

} else if (this.gameMode === 'symbol-valency') {

tileData.push({type: 'symbol', value: element.symbol, pair: element.valency.toString(), element: element});

tileData.push({type: 'valency', value: element.valency.toString(), pair: element.symbol, element: element});

} else {

tileData.push({type: 'symbol', value: element.symbol, pair: element.latin, element: element});

tileData.push({type: 'latin', value: element.latin, pair: element.symbol, element: element});

}

});

tileData = this.shuffleArray(tileData);

for (let row = 0; row < gridSize; row++) {

for (let col = 0; col < gridSize; col++) {

const index = row \* gridSize + col;

const data = tileData[index];

const x = startX + col \* (tileWidth + spacing);

const y = startY + row \* (tileHeight + spacing);

const container = new PIXI.Container();

container.x = x;

container.y = y;

container.data = data;

container.index = index;

container.visible = true;

const tile = createSprite('element\_tile\_unselected', {width: tileWidth});

tile.anchor.set(0.5);

tile.x = tileWidth / 2;

tile.y = tileHeight / 2;

const text = createText(data.value, {

fontSize: 20,

fill: 0x333333,

fontWeight: 'bold'

}, {x: tileWidth / 2, y: tileHeight / 2, anchor: 0.5});

container.addChild(tile);

container.addChild(text);

container.interactive = true;

container.buttonMode = true;

container.on('pointerdown', () => this.onTileClick(container));

app.stage.addChild(container);

this.tiles.push({

container: container,

data: data,

tile: tile,

text: text

});

}

}

}

onTileClick(tileContainer) {

if (this.gameOver) return;

const tileIndex = this.selectedTiles.indexOf(tileContainer);

if (tileIndex !== -1) {

tileContainer.children[0].texture = PIXI.Texture.from(getAssetUrl('element\_tile\_unselected'));

this.selectedTiles.splice(tileIndex, 1);

return;

}

if (this.selectedTiles.length < 2) {

tileContainer.children[0].texture = PIXI.Texture.from(getAssetUrl('element\_tile\_selected'));

this.selectedTiles.push(tileContainer);

if (this.selectedTiles.length === 2) {

this.checkMatch();

}

}

}

checkMatch() {

const tile1 = this.selectedTiles[0];

const tile2 = this.selectedTiles[1];

const isMatch = tile1.data.value === tile2.data.pair || tile2.data.value === tile1.data.pair;

const currentTime = Date.now();

if (isMatch) {

this.animateTileMatch(tile1);

this.animateTileMatch(tile2);

const timeSinceLastMatch = currentTime - this.lastMatchTime;

if (timeSinceLastMatch < 3000) {

this.combo++;

} else {

this.combo = 1;

}

this.lastMatchTime = currentTime;

const baseScore = 100;

const comboBonus = this.combo > 1 ? (this.combo - 1) \* 50 : 0;

const totalScore = baseScore + comboBonus;

this.score += totalScore;

this.scoreText.text = `Score: ${this.score}`;

this.comboText.text = `Combo: x${this.combo}`;

this.createScorePopup(tile1.container.x + 60, tile1.container.y + 45, `+${totalScore}`);

this.createParticles(tile1.container.x + 60, tile1.container.y + 45);

setTimeout(() => {

tile1.visible = false;

tile2.visible = false;

const remainingTiles = this.tiles.filter(t => t.container.visible);

if (remainingTiles.length === 0) {

this.levelComplete();

}

}, 300);

} else {

tile1.children[0].texture = PIXI.Texture.from(getAssetUrl('element\_tile\_unselected'));

tile2.children[0].texture = PIXI.Texture.from(getAssetUrl('element\_tile\_unselected'));

this.timeLeft -= 2;

this.combo = 0;

this.comboText.text = 'Combo: x1';

this.animateWrongMatch(tile1);

this.animateWrongMatch(tile2);

}

this.selectedTiles = [];

}

createParticles(x, y) {

for (let i = 0; i < 10; i++) {

const particle = new PIXI.Graphics();

particle.beginFill(0xFFD700);

particle.drawCircle(0, 0, 3);

particle.endFill();

particle.x = x;

particle.y = y;

particle.vx = (Math.random() - 0.5) \* 8;

particle.vy = (Math.random() - 0.5) \* 8;

particle.life = 1;

app.stage.addChild(particle);

this.particles.push(particle);

}

}

updateParticles(delta) {

for (let i = this.particles.length - 1; i >= 0; i--) {

const particle = this.particles[i];

particle.x += particle.vx;

particle.y += particle.vy;

particle.vy += 0.2;

particle.life -= delta / 100;

particle.alpha = particle.life;

if (particle.life <= 0) {

app.stage.removeChild(particle);

this.particles.splice(i, 1);

}

}

}

createScorePopup(x, y, text) {

const popup = createText(text, {

fontSize: 24,

fill: 0x00FF00,

fontWeight: 'bold'

}, {x: x, y: y, anchor: 0.5});

app.stage.addChild(popup);

this.scorePopups.push({

text: popup,

y: popup.y,

life: 1

});

}

updateScorePopups(delta) {

for (let i = this.scorePopups.length - 1; i >= 0; i--) {

const popup = this.scorePopups[i];

popup.text.y -= 1;

popup.life -= delta / 100;

popup.text.alpha = popup.life;

if (popup.life <= 0) {

app.stage.removeChild(popup.text);

this.scorePopups.splice(i, 1);

}

}

}

animateTileMatch(tile) {

tile.container.scale.set(1.2);

const animate = () => {

tile.container.scale.x = Math.max(1, tile.container.scale.x - 0.02);

tile.container.scale.y = Math.max(1, tile.container.scale.y - 0.02);

if (tile.container.scale.x > 1) {

requestAnimationFrame(animate);

}

};

animate();

}

animateWrongMatch(tile) {

tile.container.scale.set(0.9);

tile.container.rotation = 0.1;

const animate = () => {

tile.container.scale.x = Math.min(1, tile.container.scale.x + 0.02);

tile.container.scale.y = Math.min(1, tile.container.scale.y + 0.02);

tile.container.rotation \*= 0.9;

if (Math.abs(tile.container.rotation) > 0.01) {

requestAnimationFrame(animate);

} else {

tile.container.rotation = 0;

}

};

animate();

}

updateModeDisplay() {

const modeNames = {

'symbol-name': 'Symbol ↔ Name',

'number-symbol': 'Number ↔ Symbol',

'symbol-valency': 'Symbol ↔ Valency',

'symbol-latin': 'Symbol ↔ Latin'

};

this.modeText.text = `Mode: ${modeNames[this.gameMode]}`;

}

levelComplete() {

this.level++;

this.gameMode = ['symbol-name', 'number-symbol', 'symbol-valency', 'symbol-latin'][this.level % 4];

this.timeLeft = Math.min(this.timeLeft + 30, 90);

this.levelText.text = `Level: ${this.level}`;

this.updateModeDisplay();

this.tiles.forEach(tileObj => {

tileObj.container.destroy();

});

this.tiles = [];

this.selectedTiles = [];

this.createGrid();

}

update(delta) {

if (this.gameOver) return;

this.lastTime += delta;

if (this.lastTime >= 100) {

this.lastTime = 0;

this.timeLeft--;

this.timerText.text = `Time: ${this.timeLeft}`;

if (this.timeLeft <= 10) {

this.timerText.style.fill = 0xFF0000;

} else {

this.timerText.style.fill = 0xFFFFFF;

}

if (this.timeLeft <= 0) {

this.endGame();

}

}

this.updateParticles(delta);

this.updateScorePopups(delta);

}

endGame() {

this.gameOver = true;

showGameOver(app, {

title: 'Game Over!',

score: `Final Score: ${this.score}`,

onRestart: () => this.restart()

});

}

restart() {

app.stage.removeChildren();

this.score = 0;

this.timeLeft = 60;

this.level = 1;

this.gameOver = false;

this.lastTime = 0;

this.tiles = [];

this.selectedTiles = [];

this.setup();

}

shuffleArray(array) {

const shuffled = [...array];

for (let i = shuffled.length - 1; i > 0; i--) {

const j = Math.floor(Math.random() \* (i + 1));

[shuffled[i], shuffled[j]] = [shuffled[j], shuffled[i]];

}

return shuffled;

}

}