|  |  |
| --- | --- |
| **Button** | |
| typedef struct {  float x, y;  float width, height;  const char\* text; } Button; | typedef struct {  float x, y;  float width, height;  const char\* text;  **float textOffsetX; // новый: сдвиг текста по X**  **float textOffsetY; // новый: сдвиг текста по Y** } Button; |
| • Добавлены поля: textOffsetX, textOffsetY | |
| **EnemyTank** | |
| typedef struct {  float x, y; // Позиция центра  int direction; // Направление (0-3)  bool active; // Активен ли танк  int lastShotTime; // Время последнего выстрела  // Поля для управления движением  bool isMoving;  float moveProgress;  int targetX, targetY; // Целевая позиция в клетках  **int attackLineType; // 0: y=2, 1: x=17, 2: x=18**  **int targetLineX, targetLineY; // Конкретная точка на линии атаки**  int pathStep; // Текущий шаг в пути  int pathLength; // Длина пути  int\* path; // Массив точек пути (x0,y0,x1,y1,...)  int moveDelay; // Задержка между движениями  int lastMoveTime; // Время последнего движения  int state; // 0: к линии атаки, 1: на линии атаки } EnemyTank; | typedef struct {  float x, y; // Позиция центра  int direction; // Направление (0-3)  bool active; // Активен ли танк  int lastShotTime; // Время последнего выстрела  // Поля для управления движением  bool isMoving;  float moveProgress;  int targetX, targetY; // Целевая позиция в клетках  int pathStep; // Текущий шаг в пути  int pathLength; // Длина пути  int\* path; // Массив точек пути (x0,y0,x1,y1,...)  int moveDelay; // Задержка между движениями  int lastMoveTime; // Время последнего движения  int state; // 0: к линии атаки, 1: на линии атаки } EnemyTank; |
| • Удалены поля: attackLineType, targetLineX, targetLineY | |
| **createBullet** | |
| void createBullet() {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);   // Проверяем задержку между выстрелами  if (currentTime - lastShotTime < SHOT\_DELAY) return;   // Ищем свободный слот для пули  for (int i = 0; i < MAX\_BULLETS; i++) {  if (!bullets[i].active) {  bullets[i].x = playerTank.x + (playerTank.direction - 2) % 2 \* 0.5f;  bullets[i].y = playerTank.y + (playerTank.direction - 1) % 2 \* 0.5f;  bullets[i].direction = playerTank.direction;  bullets[i].active = true;  bullets[i].isEnemy = false; // Помечаем как пулю игрока   // Корректируем позицию в зависимости от направления  switch (playerTank.direction) {  case 0: bullets[i].y += 0.6f; break; // Вверх  **case 1: bullets[i].x -= 0.6f; break; // Влево**  case 2: bullets[i].y -= 0.6f; break; // Вниз  **case 3: bullets[i].x += 0.6f; break; // Вправо**  }   lastShotTime = currentTime;  return;  }  } } | void createBullet() {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);   // Проверяем задержку между выстрелами  if (currentTime - lastShotTime < SHOT\_DELAY) return;   // Ищем свободный слот для пули  for (int i = 0; i < MAX\_BULLETS; i++) {  if (!bullets[i].active) {  bullets[i].x = playerTank.x + (playerTank.direction - 2) % 2 \* 0.5f;  bullets[i].y = playerTank.y + (playerTank.direction - 1) % 2 \* 0.5f;  bullets[i].direction = playerTank.direction;  bullets[i].active = true;  bullets[i].isEnemy = false; // Помечаем как пулю игрока   // Корректируем позицию в зависимости от направления  switch (playerTank.direction) {  case 0: bullets[i].y += 0.6f; break; // Вверх  **case 1: bullets[i].x += 0.6f; break; // Влево**  case 2: bullets[i].y -= 0.6f; break; // Вниз  **case 3: bullets[i].x -= 0.6f; break; // Вправо**  }   lastShotTime = currentTime;  return;  }  } } |
| • Добавлены присваивания/обновления: case 1: bullets[i].x += 0.6f; break;; case 3: bullets[i].x -= 0.6f; break; | |
| **createEnemyBullet** | |
| void createEnemyBullet(int enemyIndex) {  for (int i = 0; i < MAX\_BULLETS; i++) {  if (!bullets[i].active) {  bullets[i].x = enemies[enemyIndex].x;  bullets[i].y = enemies[enemyIndex].y;  bullets[i].direction = enemies[enemyIndex].direction;  bullets[i].active = true;  bullets[i].isEnemy = true; // Помечаем как вражескую   // Корректировка позиции  switch (enemies[enemyIndex].direction) {  case 0: bullets[i].y += 0.6f; break;  **case 1: bullets[i].x += 0.6f; break;**  case 2: bullets[i].y -= 0.6f; break;  **case 3: bullets[i].x -= 0.6f; break;**  }  break;  }  } } | void createEnemyBullet(int enemyIndex) {  for (int i = 0; i < MAX\_BULLETS; i++) {  if (!bullets[i].active) {  bullets[i].x = enemies[enemyIndex].x;  bullets[i].y = enemies[enemyIndex].y;  bullets[i].direction = enemies[enemyIndex].direction;  bullets[i].active = true;  bullets[i].isEnemy = true; // Помечаем как вражескую   // Корректировка позиции  switch (enemies[enemyIndex].direction) {  case 0: bullets[i].y += 0.6f; break;  **case 1: bullets[i].x -= 0.6f; break;**  case 2: bullets[i].y -= 0.6f; break;  **case 3: bullets[i].x += 0.6f; break;**  }  break;  }  } } |
| • Добавлены присваивания/обновления: case 1: bullets[i].x -= 0.6f; break;; case 3: bullets[i].x += 0.6f; break; | |
| **display** | |
| void display() {  // Очищаем буфер цвета  glClear(GL\_COLOR\_BUFFER\_BIT);   // В зависимости от состояния игры рисуем разные экраны  switch (gameState) {  case GAME\_MENU:  drawMenu();  break;   case GAME\_PLAYING:  // Рисуем игровое поле  drawField();  drawWalls();  drawGameObjects();  drawInfoPanel();   // Отрисовка игровых объектов  updateTank();  drawTank();   respawnEnemies();  updateEnemies();  drawEnemies();   updateBullets();  drawBullets();  break;   case GAME\_WIN:  // Рисуем игровое поле на заднем плане  drawField();  drawWalls();  drawGameObjects();  drawInfoPanel();   // Отрисовка игровых объектов  drawTank();  drawEnemies();  drawBullets();   // Рисуем экран победы поверх  drawWinScreen();  break;   case GAME\_LOSE:  // Рисуем игровое поле на заднем плане  drawField();  drawWalls();  drawGameObjects();  drawInfoPanel();   // Отрисовка игровых объектов  drawTank();  drawEnemies();  drawBullets();   // Рисуем экран проигрыша поверх  drawLoseScreen();  break;  }   // Обмен буферов  glutSwapBuffers();  glutPostRedisplay(); // Непрерывная перерисовка } | void display() {  // Очищаем буфер цвета  glClear(GL\_COLOR\_BUFFER\_BIT);    **// фэпэсы считаем**  **int nowMs = glutGet(GLUT\_ELAPSED\_TIME);**  **if (fpsLastTimeMs == 0) fpsLastTimeMs = nowMs; // инициализация при первом кадре**  **fpsFrameCount++;**  **int delta = nowMs - fpsLastTimeMs;**  **if (delta >= 1000) { // раз в ~секунду**  **fps = (int)((fpsFrameCount \* 1000.0f) / (float)delta);**  **fpsFrameCount = 0;**  **fpsLastTimeMs = nowMs;**  **}**    // В зависимости от состояния игры рисуем разные экраны  switch (gameState) {  case GAME\_MENU:  drawMenu();  break;   case GAME\_PLAYING:  // Рисуем игровое поле  drawField();  drawWalls();  drawGameObjects();  drawInfoPanel();   // Отрисовка игровых объектов  updateTank();  drawTank();   respawnEnemies();  updateEnemies();  drawEnemies();   updateBullets();  drawBullets();  break;   case GAME\_WIN:  // Рисуем игровое поле на заднем плане  drawField();  drawWalls();  drawGameObjects();  drawInfoPanel();   // Отрисовка игровых объектов  drawTank();  drawEnemies();  drawBullets();   // Рисуем экран победы поверх  drawWinScreen();  break;   case GAME\_LOSE:  // Рисуем игровое поле на заднем плане  drawField();  drawWalls();  drawGameObjects();  drawInfoPanel();   // Отрисовка игровых объектов  drawTank();  drawEnemies();  drawBullets();   // Рисуем экран проигрыша поверх  drawLoseScreen();  break;  }   // Обмен буферов  glutSwapBuffers();  glutPostRedisplay(); // Непрерывная перерисовка } |
| • Добавлены локальные переменные: fpsFrameCount++ • Добавлены вызовы функций: glutGet • Добавлены условия: if (fpsLastTimeMs == 0) fpsLastTimeMs = nowMs;; if (delta >= 1000) { • Добавлены присваивания/обновления: int nowMs = glutGet(GLUT\_ELAPSED\_TIME);; int delta = nowMs - fpsLastTimeMs;; fps = (int)((fpsFrameCount \* 1000.0f) / (float)delta);; fpsFrameCount = 0;; fpsLastTimeMs = nowMs; | |
| **drawButton** | |
| void drawButton(Button button) {  // Рисуем прямоугольник кнопки  **glColor3f(0.3f, 0.2f, 0.1f); // Оранжевый**  glBegin(GL\_QUADS);  glVertex2f(button.x, button.y);  glVertex2f(button.x + button.width, button.y);  glVertex2f(button.x + button.width, button.y + button.height);  glVertex2f(button.x, button.y + button.height);  glEnd();   // Рамка кнопки  **glColor3f(0.7f, 0.3f, 0.1f); // Темно-оранжевый**  glLineWidth(2.0f);  glBegin(GL\_LINE\_LOOP);  glVertex2f(button.x, button.y);  glVertex2f(button.x + button.width, button.y);  glVertex2f(button.x + button.width, button.y + button.height);  glVertex2f(button.x, button.y + button.height);  glEnd();  glLineWidth(1.0f);   **// Центрирование текста**  int textWidth = 0;  for (const char\* c = button.text; \*c != '\0'; c++) {  textWidth += glutBitmapWidth(GLUT\_BITMAP\_9\_BY\_15, \*c);  }   **float textX = button.x + (button.width - textWidth / 100.0f) / 2;**  **float textY = button.y + button.height / 2 - 0.15f;**   **drawText(textX - 2.0, textY, button.text);** } | void drawButton(Button button) {  // Рисуем прямоугольник кнопки  **glColor3f(0.44f, 0.50f, 0.56f); // Основной цвет кнопки (#708090)**  glBegin(GL\_QUADS);  glVertex2f(button.x, button.y);  glVertex2f(button.x + button.width, button.y);  glVertex2f(button.x + button.width, button.y + button.height);  glVertex2f(button.x, button.y + button.height);  glEnd();   // Рамка кнопки  **glColor3f(0.69f, 0.77f, 0.87f); // Обводка кнопки (#B0C4DE)**  glLineWidth(2.0f);  glBegin(GL\_LINE\_LOOP);  glVertex2f(button.x, button.y);  glVertex2f(button.x + button.width, button.y);  glVertex2f(button.x + button.width, button.y + button.height);  glVertex2f(button.x, button.y + button.height);  glEnd();  glLineWidth(1.0f);   **// Центрирование текста + учёт индивидуальных сдвигов**  int textWidth = 0;  for (const char\* c = button.text; \*c != '\0'; c++) {  textWidth += glutBitmapWidth(GLUT\_BITMAP\_9\_BY\_15, \*c);  }  **float textW = textWidth / 100.0f; // перевод «пикселей» шрифта в мир**  **float textX = button.x + (button.width - textW) \* 0.5f; // центр по X**  **float textY = button.y + (button.height \* 0.5f) - 0.15f; // центр по Y, чуть ниже**   **drawText(textX + button.textOffsetX, textY + button.textOffsetY, button.text);**  } |
| • Добавлены вызовы функций: drawText, glColor3f • Добавлены присваивания/обновления: float textW = textWidth / 100.0f;; float textX = button.x + (button.width - textW) \* 0.5f;; float textY = button.y + (button.height \* 0.5f) - 0.15f; | |
| **drawEnemyTank** | |
| void drawEnemyTank(EnemyTank\* tank) {  glPushMatrix();  glTranslatef(tank->x, tank->y, 0.0f);   // Всегда используем актуальное направление из структуры  switch (tank->direction) {  **case 0: break; // Вверх**  **case 1: glRotatef(90.0f, 0, 0, 1); break; // Влево**  **case 2: glRotatef(180.0f, 0, 0, 1); break; // Вниз**  **case 3: glRotatef(270.0f, 0, 0, 1); break; // Вправо**  }   **// Основной цвет - бронзовый**  **glColor3f(0.8f, 0.5f, 0.2f);**   // Корпус  glBegin(GL\_QUADS);  glVertex2f(-0.5f, -0.5f);  glVertex2f(0.5f, -0.5f);  glVertex2f(0.5f, 0.5f);  glVertex2f(-0.5f, 0.5f);  glEnd();   // Пушка  glColor3f(0.3f, 0.3f, 0.3f);  glBegin(GL\_QUADS);  glVertex2f(-0.05f, 0.0f);  glVertex2f(0.05f, 0.0f);  glVertex2f(0.05f, 0.6f);  glVertex2f(-0.05f, 0.6f);  glEnd();   // Гусеницы  glColor3f(0.4f, 0.3f, 0.2f);  glBegin(GL\_QUADS);  // Левая  glVertex2f(-0.6f, -0.4f);  glVertex2f(-0.4f, -0.4f);  glVertex2f(-0.4f, 0.4f);  glVertex2f(-0.6f, 0.4f);  // Правая  glVertex2f(0.4f, -0.4f);  glVertex2f(0.6f, -0.4f);  glVertex2f(0.6f, 0.4f);  glVertex2f(0.4f, 0.4f);  glEnd();   glPopMatrix(); } | void drawEnemyTank(EnemyTank\* tank) {  glPushMatrix();  glTranslatef(tank->x, tank->y, 0.0f);   // Всегда используем актуальное направление из структуры  switch (tank->direction) {  **case 0: break;**  **case 1: glRotatef(90.0f, 0, 0, 1); break; // лево**  **case 2: glRotatef(180.0f, 0, 0, 1); break; // низ**  **case 3: glRotatef(-90.0f, 0, 0, 1); break; // право**  }   **glColor3f(1.0f, 0.85f, 0.73f); // Новый цвет врага (#FFDAB9)**    // Корпус  glBegin(GL\_QUADS);  glVertex2f(-0.5f, -0.5f);  glVertex2f(0.5f, -0.5f);  glVertex2f(0.5f, 0.5f);  glVertex2f(-0.5f, 0.5f);  glEnd();   // Пушка  glColor3f(0.3f, 0.3f, 0.3f);  glBegin(GL\_QUADS);  glVertex2f(-0.05f, 0.0f);  glVertex2f(0.05f, 0.0f);  glVertex2f(0.05f, 0.6f);  glVertex2f(-0.05f, 0.6f);  glEnd();   // Гусеницы  glColor3f(0.4f, 0.3f, 0.2f);  glBegin(GL\_QUADS);  // Левая  glVertex2f(-0.6f, -0.4f);  glVertex2f(-0.4f, -0.4f);  glVertex2f(-0.4f, 0.4f);  glVertex2f(-0.6f, 0.4f);  // Правая  glVertex2f(0.4f, -0.4f);  glVertex2f(0.6f, -0.4f);  glVertex2f(0.6f, 0.4f);  glVertex2f(0.4f, 0.4f);  glEnd();   glPopMatrix(); } |
| • Добавлены локальные переменные: break • Добавлены вызовы функций: glColor3f, glRotatef | |
| **drawField** | |
| void drawField() {  // Рисуем игровое поле (зеленый квадрат)  **glColor3f(0.2f, 0.4f, 0.1f); // Зеленый цвет поля**  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glEnd(); } | void drawField() {  // Рисуем игровое поле (зеленый квадрат)  **glColor3f(0.36f, 0.54f, 0.45f); // #5B8A72 — серо-зелёный металл**  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glEnd(); } |
| • Добавлены вызовы функций: glColor3f | |
| **drawGameObjects** | |
| void drawGameObjects() {  // Рассчитываем начальные координаты (левый нижний угол сетки)  float startX = -FIELD\_WIDTH / 2 + WALL\_THICKNESS;  float startY = -FIELD\_HEIGHT / 2 + WALL\_THICKNESS;   // Проходим по всем ячейкам сетки  for (int y = 0; y < GRID\_HEIGHT; y++) {  for (int x = 0; x < GRID\_WIDTH; x++) {  float posX = startX + x \* CELL\_SIZE;  float posY = startY + y \* CELL\_SIZE;   switch (gameGrid[y][x].type) {  case OBJECT\_BRICK: // Разрушаемая стена (кирпич)  // Определяем цвет по прочности  switch (gameGrid[y][x].durability) {  **case 3: glColor3f(0.8f, 0.1f, 0.1f); break; // Темно-красный**  **case 2: glColor3f(0.9f, 0.3f, 0.3f); break; // Средне-красный**  **case 1: glColor3f(1.0f, 0.5f, 0.5f); break; // Светло-красный**  }  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;   case OBJECT\_BRICK\_BASE: // Разрушаемая стена базы (кирпич)  // Определяем цвет по прочности  switch (gameGrid[y][x].durability) {  **case 3: glColor3f(0.8f, 0.1f, 0.1f); break; // Темно-красный**  **case 2: glColor3f(0.9f, 0.3f, 0.3f); break; // Средне-красный**  **case 1: glColor3f(1.0f, 0.5f, 0.5f); break; // Светло-красный**  }  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;   case OBJECT\_STEEL: // Неразрушаемая стена (сталь)  **glColor3f(0.5f, 0.5f, 0.5f); // Серый**  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;   case OBJECT\_BASE: // База игрока  **glColor3f(0.9f, 0.9f, 0.2f); // Желтый**  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;  }  }  } } | void drawGameObjects() {  // Рассчитываем начальные координаты (левый нижний угол сетки)  float startX = -FIELD\_WIDTH / 2 + WALL\_THICKNESS;  float startY = -FIELD\_HEIGHT / 2 + WALL\_THICKNESS;   // Проходим по всем ячейкам сетки  for (int y = 0; y < GRID\_HEIGHT; y++) {  for (int x = 0; x < GRID\_WIDTH; x++) {  float posX = startX + x \* CELL\_SIZE;  float posY = startY + y \* CELL\_SIZE;   switch (gameGrid[y][x].type) {  case OBJECT\_BRICK: // Разрушаемая стена (кирпич)  // Определяем цвет по прочности  switch (gameGrid[y][x].durability) {  **case 3: glColor3f(0.80f, 0.44f, 0.44f); break; // основа**  **case 2: glColor3f(0.88f, 0.56f, 0.56f); break; // средне**  **case 1: glColor3f(0.96f, 0.68f, 0.68f); break; // почти всё**  }  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;   case OBJECT\_BRICK\_BASE: // Разрушаемая стена базы (кирпич)  // Определяем цвет по прочности  switch (gameGrid[y][x].durability) {  **case 3: glColor3f(0.80f, 0.44f, 0.44f); break; // основа**  **case 2: glColor3f(0.88f, 0.56f, 0.56f); break; // средне**  **case 1: glColor3f(0.96f, 0.68f, 0.68f); break; // почти всё**  }  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;   case OBJECT\_STEEL: // Неразрушаемая стена (сталь)  **glColor3f(0.69f, 0.77f, 0.87f); // Цвет танка игрока (#B0C4DE)**  glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;   case OBJECT\_BASE: // База игрока  **glColor3f(0.80f, 0.67f, 0.49f); // #CDAA7D — бронзово-песочный**   glBegin(GL\_QUADS);  glVertex2f(posX, posY);  glVertex2f(posX + CELL\_SIZE, posY);  glVertex2f(posX + CELL\_SIZE, posY + CELL\_SIZE);  glVertex2f(posX, posY + CELL\_SIZE);  glEnd();  break;  }  }  } } |
| • Добавлены вызовы функций: glColor3f | |
| **drawInfoPanel** | |
| void drawInfoPanel() {  // Рисуем панель  float panelLeft = FIELD\_WIDTH / 2;  float panelRight = panelLeft + PANEL\_WIDTH;  float panelBottom = -PANEL\_HEIGHT / 2;  float panelTop = PANEL\_HEIGHT / 2;   **glColor3f(0.9f, 0.5f, 0.1f); // Оранжевый**  glBegin(GL\_QUADS);  glVertex2f(panelLeft, panelBottom);  glVertex2f(panelRight, panelBottom);  glVertex2f(panelRight, panelTop);  glVertex2f(panelLeft, panelTop);  glEnd();   // Рисуем информацию  glColor3f(0.0f, 0.0f, 0.0f); // Черный   // Здоровье игрока  char hpText[20];  snprintf(hpText, sizeof(hpText), "HP: %d", playerHP);  drawText(panelLeft + 0.5f, panelTop - 1.0f, hpText);   // Время игры  int minutes = gameTime / 60;  int seconds = gameTime % 60;  char timeText[20];  snprintf(timeText, sizeof(timeText), "Time: %02d:%02d", minutes, seconds);  drawText(panelLeft + 0.5f, panelTop - 2.5f, timeText);   // Очки  char scoreText[20];  snprintf(scoreText, sizeof(scoreText), "Score: %d", gameScore);  drawText(panelLeft + 0.5f, panelTop - 4.0f, scoreText); } | void drawInfoPanel() {  // Рисуем панель  float panelLeft = FIELD\_WIDTH / 2;  float panelRight = panelLeft + PANEL\_WIDTH;  float panelBottom = -PANEL\_HEIGHT / 2;  float panelTop = PANEL\_HEIGHT / 2;   **glColor3f(0.29f, 0.33f, 0.39f); // Серо-голубой фон (#4A5563)**  glBegin(GL\_QUADS);  glVertex2f(panelLeft, panelBottom);  glVertex2f(panelRight, panelBottom);  glVertex2f(panelRight, panelTop);  glVertex2f(panelLeft, panelTop);  glEnd();   // Рисуем информацию  glColor3f(0.0f, 0.0f, 0.0f); // Черный   // Здоровье игрока  char hpText[20];  snprintf(hpText, sizeof(hpText), "HP: %d", playerHP);  drawText(panelLeft + 0.5f, panelTop - 1.0f, hpText);   // Время игры  int minutes = gameTime / 60;  int seconds = gameTime % 60;  char timeText[20];  snprintf(timeText, sizeof(timeText), "Time: %02d:%02d", minutes, seconds);  drawText(panelLeft + 0.5f, panelTop - 2.5f, timeText);   // Очки  char scoreText[20];  snprintf(scoreText, sizeof(scoreText), "Score: %d", gameScore);  drawText(panelLeft + 0.5f, panelTop - 4.0f, scoreText);   **// FPS**  **char fpsText[20];**  **snprintf(fpsText, sizeof(fpsText), "FPS: %d", fps);**  **// Тот же левый отступ, на строку ниже очков (шаг 1.5 как у остальных строк)**  **drawText(panelLeft + 0.5f, panelTop - 5.5f, fpsText);**  } |
| • Добавлены локальные переменные: fpsText • Добавлены вызовы функций: drawText, glColor3f, snprintf | |
| **drawLoseScreen** | |
| void drawLoseScreen() {  **// Полупрозрачный черный фон**  **glColor4f(0.0f, 0.0f, 0.0f, 0.7f);**  glBegin(GL\_QUADS);  **glVertex2f(-10.0f, -10.0f);**  **glVertex2f(10.0f, -10.0f);**  **glVertex2f(10.0f, 10.0f);**  **glVertex2f(-10.0f, 10.0f);**  glEnd();   // Текст проигрыша  glColor3f(1.0f, 0.0f, 0.0f); // Красный  drawText(-3.0f, 2.0f, "You loooose!");   // Причина проигрыша  if (playerHP <= 0) {  drawText(-3.0f, 0.5f, "You died");  }  else {  drawText(-3.0f, 0.5f, "Base was destroyed");  }   // Очки  char scoreText[30];  sprintf(scoreText, "Your score: %d", gameScore);  drawText(-3.0f, -1.0f, scoreText);   **// Кнопка OK**  **drawButton(okButton);** } | void drawLoseScreen() {  **glColor3f(0.35f, 0.40f, 0.46f); // фон панели в цвет фона**   glBegin(GL\_QUADS);  **glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);**  **glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);**  **glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);**  **glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);**  glEnd();    // Текст проигрыша  glColor3f(1.0f, 0.0f, 0.0f); // Красный  drawText(-3.0f, 2.0f, "You loooose!");   // Причина проигрыша  if (playerHP <= 0) {  drawText(-3.0f, 0.5f, "You died");  }  else {  drawText(-3.0f, 0.5f, "Base was destroyed");  }   // Очки  char scoreText[30];  sprintf(scoreText, "Your score: %d", gameScore);  drawText(-3.0f, -1.0f, scoreText);   **// Кнопки Game Over и Restart**  **drawButton(gameOverButton);**  **drawButton(restartButton);** } |
| • Добавлены вызовы функций: drawButton, glColor3f, glVertex2f | |
| **drawMenu** | |
| void drawMenu() {  **// Очищаем экран оранжевым цветом**  **glClearColor(0.9f, 0.5f, 0.1f, 1.0f);**  glClear(GL\_COLOR\_BUFFER\_BIT);   // Рисуем заголовок  glColor3f(0.0f, 0.0f, 0.0f);  **drawText(-4.0f, 4.0f, "BATTLE GYM");**   // Рисуем кнопки  drawButton(startButton);  drawButton(exitButton); } | void drawMenu() {  **glClearColor(0.29f, 0.33f, 0.39f, 1.0f); // Новый серо-голубой фон (#4A5563)**  glClear(GL\_COLOR\_BUFFER\_BIT);   // Рисуем заголовок  glColor3f(0.0f, 0.0f, 0.0f);  **drawText(1.31f, 3.0f, "BATTLE GYM");**   // Рисуем кнопки  drawButton(startButton);  drawButton(exitButton); } |
| • Добавлены вызовы функций: drawText, glClearColor | |
| **drawTank** | |
| void drawTank() {  glPushMatrix();  glTranslatef(playerTank.x, playerTank.y, 0.0f);   // Поворот в зависимости от направления  switch (playerTank.direction) {  case 0: break; // Вверх  case 1: glRotatef(90.0f, 0, 0, 1); break;  case 2: glRotatef(180.0f, 0, 0, 1); break;  case 3: glRotatef(270.0f, 0, 0, 1); break;  }   **// Основной цвет - песочный**  **glColor3f(0.76f, 0.70f, 0.50f);**   // Корпус (основной прямоугольник)  glBegin(GL\_QUADS);  glVertex2f(-0.5f, -0.5f);  glVertex2f(0.5f, -0.5f);  glVertex2f(0.5f, 0.5f);  glVertex2f(-0.5f, 0.5f);  glEnd();   // Пушка  glColor3f(0.3f, 0.3f, 0.3f);  glBegin(GL\_QUADS);  glVertex2f(-0.05f, 0.0f);  glVertex2f(0.05f, 0.0f);  glVertex2f(0.05f, 0.6f);  glVertex2f(-0.05f, 0.6f);  glEnd();   // Гусеницы  glColor3f(0.4f, 0.4f, 0.4f);  glBegin(GL\_QUADS);  // Левая  glVertex2f(-0.6f, -0.4f);  glVertex2f(-0.4f, -0.4f);  glVertex2f(-0.4f, 0.4f);  glVertex2f(-0.6f, 0.4f);   // Правая  glVertex2f(0.4f, -0.4f);  glVertex2f(0.6f, -0.4f);  glVertex2f(0.6f, 0.4f);  glVertex2f(0.4f, 0.4f);  glEnd();   glPopMatrix(); } | void drawTank() {  glPushMatrix();  glTranslatef(playerTank.x, playerTank.y, 0.0f);   // Поворот в зависимости от направления  switch (playerTank.direction) {  case 0: break; // Вверх  case 1: glRotatef(90.0f, 0, 0, 1); break;  case 2: glRotatef(180.0f, 0, 0, 1); break;  case 3: glRotatef(270.0f, 0, 0, 1); break;  }   **glColor3f(0.69f, 0.77f, 0.87f); // Новый цвет игрока (#B0C4DE)**    // Корпус (основной прямоугольник)  glBegin(GL\_QUADS);  glVertex2f(-0.5f, -0.5f);  glVertex2f(0.5f, -0.5f);  glVertex2f(0.5f, 0.5f);  glVertex2f(-0.5f, 0.5f);  glEnd();   // Пушка  glColor3f(0.3f, 0.3f, 0.3f);  glBegin(GL\_QUADS);  glVertex2f(-0.05f, 0.0f);  glVertex2f(0.05f, 0.0f);  glVertex2f(0.05f, 0.6f);  glVertex2f(-0.05f, 0.6f);  glEnd();   // Гусеницы  glColor3f(0.4f, 0.4f, 0.4f);  glBegin(GL\_QUADS);  // Левая  glVertex2f(-0.6f, -0.4f);  glVertex2f(-0.4f, -0.4f);  glVertex2f(-0.4f, 0.4f);  glVertex2f(-0.6f, 0.4f);   // Правая  glVertex2f(0.4f, -0.4f);  glVertex2f(0.6f, -0.4f);  glVertex2f(0.6f, 0.4f);  glVertex2f(0.4f, 0.4f);  glEnd();   glPopMatrix(); } |
| • Добавлены вызовы функций: glColor3f | |
| **drawWalls** | |
| void drawWalls() {  **// Устанавливаем оранжевый цвет для стен**  **glColor3f(0.9f, 0.5f, 0.1f);**   // Левая стена  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2 + WALL\_THICKNESS, -FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2 + WALL\_THICKNESS, FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glEnd();   // Правая стена  glBegin(GL\_QUADS);  glVertex2f(FIELD\_WIDTH / 2 - WALL\_THICKNESS, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2 - WALL\_THICKNESS, FIELD\_HEIGHT / 2);  glEnd();   // Верхняя стена  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2 - WALL\_THICKNESS);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2 - WALL\_THICKNESS);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glEnd();   // Нижняя стена  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2 + WALL\_THICKNESS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2 + WALL\_THICKNESS);  glEnd(); } | void drawWalls() {  **glColor3f(0.44f, 0.50f, 0.56f); // Тёмно-серый (#708090)**      // Левая стена  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2 + WALL\_THICKNESS, -FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2 + WALL\_THICKNESS, FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glEnd();   // Правая стена  glBegin(GL\_QUADS);  glVertex2f(FIELD\_WIDTH / 2 - WALL\_THICKNESS, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2 - WALL\_THICKNESS, FIELD\_HEIGHT / 2);  glEnd();   // Верхняя стена  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2 - WALL\_THICKNESS);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2 - WALL\_THICKNESS);  glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);  glEnd();   // Нижняя стена  glBegin(GL\_QUADS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);  glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2 + WALL\_THICKNESS);  glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2 + WALL\_THICKNESS);  glEnd(); } |
| • Добавлены вызовы функций: glColor3f | |
| **drawWinScreen** | |
| void drawWinScreen() {  **// Полупрозрачный черный фон**  **glColor4f(0.0f, 0.0f, 0.0f, 0.7f);**  glBegin(GL\_QUADS);  **glVertex2f(-10.0f, -10.0f);**  **glVertex2f(10.0f, -10.0f);**  **glVertex2f(10.0f, 10.0f);**  **glVertex2f(-10.0f, 10.0f);**  glEnd();   // Текст победы  **glColor3f(0.0f, 1.0f, 0.0f); // Зеленый**  drawText(-4.0f, 2.0f, "You win!");   // Очки  char scoreText[30];  sprintf(scoreText, "Your score: %d", gameScore);  **drawText(-4.0f, 0.0f, scoreText);**   **// Кнопка OK**  drawButton(okButton); } | void drawWinScreen() {  **glColor3f(0.35f, 0.40f, 0.46f); // #5A6675 примерно**  glBegin(GL\_QUADS);  **glVertex2f(-FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);**  **glVertex2f(FIELD\_WIDTH / 2, -FIELD\_HEIGHT / 2);**  **glVertex2f(FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);**  **glVertex2f(-FIELD\_WIDTH / 2, FIELD\_HEIGHT / 2);**  glEnd();   // Текст победы  **glColor3f(0.0f, 1.0f, 0.0f);**  drawText(-4.0f, 2.0f, "You win!");   // Очки  char scoreText[30];  sprintf(scoreText, "Your score: %d", gameScore);  **drawText(-4.0f, 0.5f, scoreText);**   drawButton(okButton); } |
| • Добавлены вызовы функций: drawText, glColor3f, glVertex2f | |
| **findPathToAttackLine** | |
| void findPathToAttackLine(EnemyTank\* tank) {  // Очищаем предыдущий путь  if (tank->path) {  free(tank->path);  tank->path = NULL;  }   // Текущая позиция в сетке  **int startX = (int)((tank->x + 8.5f) / 0.5f);**  **int startY = (int)((tank->y + 8.5f) / 0.5f);**   // Целевые точки атаки  int targets[3][2] = { {17, 1}, {18, 1}, {startX, 1} };  PathNode bestTarget = { -1, -1, -1, -1 };  bool found = false;  int shortestPath = INT\_MAX;   for (int t = 0; t < 3; t++) {  int targetX = targets[t][0];  int targetY = targets[t][1];   // Массив посещенных точек  bool visited[36][36] = { false };   // Очередь инициализации  queueStart = 0;  queueEnd = 0;   // Добавляем стартовую точку  pathQueue[queueEnd++] = (PathNode){ startX, startY, -1, -1 };  visited[startY][startX] = true;   PathNode targetNode = { -1, -1, -1, -1 };  bool pathFound = false;   while (queueStart < queueEnd) {  PathNode current = pathQueue[queueStart++];   // Проверяем, достигли ли мы цели  if (current.x == targetX && current.y == targetY) {  targetNode = current;  pathFound = true;  break;  }   // Проверяем соседние клетки  int dx[4] = { 0, 0, -1, 1 };  int dy[4] = { -1, 1, 0, 0 };   for (int i = 0; i < 4; i++) {  int nx = current.x + dx[i];  int ny = current.y + dy[i];   // Проверяем границы  if (nx < 0 || nx >= GRID\_WIDTH || ny < 0 || ny >= GRID\_HEIGHT)  continue;   // Проверяем, можно ли пройти  if (!visited[ny][nx] && canMoveTo(nx, ny)) {  visited[ny][nx] = true;  pathQueue[queueEnd++] = (PathNode){ nx, ny, current.x, current.y };  }  }  }   // Если путь найден, проверяем длину  if (pathFound) {  int length = 0;  PathNode node = targetNode;  while (node.parentX != -1) {  length++;  for (int i = 0; i < queueEnd; i++) {  if (pathQueue[i].x == node.parentX && pathQueue[i].y == node.parentY) {  node = pathQueue[i];  break;  }  }  }   if (length < shortestPath) {  shortestPath = length;  bestTarget = targetNode;  found = true;  }  }  }   // Если путь найден, сохраняем его  if (found) {  int length = shortestPath;  tank->pathLength = length;  tank->path = malloc(length \* 2 \* sizeof(int));  tank->pathStep = 0;   PathNode node = bestTarget;  int index = length - 1;  while (node.parentX != -1 && index >= 0) {  tank->path[index \* 2] = node.x;  tank->path[index \* 2 + 1] = node.y;  index--;   for (int i = 0; i < queueEnd; i++) {  if (pathQueue[i].x == node.parentX && pathQueue[i].y == node.parentY) {  node = pathQueue[i];  break;  }  }  }   // Устанавливаем первую цель пути  if (tank->pathLength > 0) {  tank->targetX = tank->path[0];  tank->targetY = tank->path[1];  }  } } | void findPathToAttackLine(EnemyTank\* tank) {  // Очищаем предыдущий путь  if (tank->path) {  free(tank->path);  tank->path = NULL;  }   // Текущая позиция в сетке  **int startX = (int)roundf((tank->x + 8.0f) / 0.5f);**  **int startY = (int)roundf((tank->y + 8.0f) / 0.5f);**   // Целевые точки атаки  int targets[3][2] = { {17, 1}, {18, 1}, {startX, 1} };  PathNode bestTarget = { -1, -1, -1, -1 };  bool found = false;  int shortestPath = INT\_MAX;   for (int t = 0; t < 3; t++) {  int targetX = targets[t][0];  int targetY = targets[t][1];   // Массив посещенных точек  bool visited[36][36] = { false };   // Очередь инициализации  queueStart = 0;  queueEnd = 0;   // Добавляем стартовую точку  pathQueue[queueEnd++] = (PathNode){ startX, startY, -1, -1 };  visited[startY][startX] = true;   PathNode targetNode = { -1, -1, -1, -1 };  bool pathFound = false;   while (queueStart < queueEnd) {  PathNode current = pathQueue[queueStart++];   // Проверяем, достигли ли мы цели  if (current.x == targetX && current.y == targetY) {  targetNode = current;  pathFound = true;  break;  }   // Проверяем соседние клетки  int dx[4] = { 0, 0, -1, 1 };  int dy[4] = { -1, 1, 0, 0 };   for (int i = 0; i < 4; i++) {  int nx = current.x + dx[i];  int ny = current.y + dy[i];   // Проверяем границы  if (nx < 0 || nx >= GRID\_WIDTH || ny < 0 || ny >= GRID\_HEIGHT)  continue;   // Проверяем, можно ли пройти  if (!visited[ny][nx] && canMoveTo(nx, ny)) {  visited[ny][nx] = true;  pathQueue[queueEnd++] = (PathNode){ nx, ny, current.x, current.y };  }  }  }   // Если путь найден, проверяем длину  if (pathFound) {  int length = 0;  PathNode node = targetNode;  while (node.parentX != -1) {  length++;  for (int i = 0; i < queueEnd; i++) {  if (pathQueue[i].x == node.parentX && pathQueue[i].y == node.parentY) {  node = pathQueue[i];  break;  }  }  }   if (length < shortestPath) {  shortestPath = length;  bestTarget = targetNode;  found = true;  }  }  }   // Если путь найден, сохраняем его  if (found) {  int length = shortestPath;  tank->pathLength = length;  tank->path = malloc(length \* 2 \* sizeof(int));  tank->pathStep = 0;   PathNode node = bestTarget;  int index = length - 1;  while (node.parentX != -1 && index >= 0) {  tank->path[index \* 2] = node.x;  tank->path[index \* 2 + 1] = node.y;  index--;   for (int i = 0; i < queueEnd; i++) {  if (pathQueue[i].x == node.parentX && pathQueue[i].y == node.parentY) {  node = pathQueue[i];  break;  }  }  }   // Устанавливаем первую цель пути  if (tank->pathLength > 0) {  tank->targetX = tank->path[0];  tank->targetY = tank->path[1];  }  } } |
| • Добавлены вызовы функций: roundf • Добавлены присваивания/обновления: int startX = (int)roundf((tank->x + 8.0f) / 0.5f);; int startY = (int)roundf((tank->y + 8.0f) / 0.5f); | |
| **getDirectionToTarget** | |
| int getDirectionToTarget(int currentX, int currentY, int targetX, int targetY) {  int dx = targetX - currentX;  int dy = targetY - currentY;   // Приоритет движения по оси с наибольшей разницей  if (abs(dx) > abs(dy)) {  **return (dx > 0) ? 3 : 1; // Вправо : Влево**  }  else {  return (dy > 0) ? 0 : 2; // Вверх : Вниз  } } | int getDirectionToTarget(int currentX, int currentY, int targetX, int targetY) {  int dx = targetX - currentX;  int dy = targetY - currentY;   // Приоритет движения по оси с наибольшей разницей  if (abs(dx) > abs(dy)) {  **return (dx > 0) ? 1 : 3; // Вправо : Влево**  }  else {  return (dy > 0) ? 0 : 2; // Вверх : Вниз  } } |
| • Обновлен код/логика без явных добавлений | |
| **init** | |
| void init() {  // Устанавливаем цвет очистки  **glClearColor(0.1f, 0.1f, 0.1f, 1.0f);**  initGame(); } | void init() {  // Устанавливаем цвет очистки  **glClearColor(0.29f, 0.33f, 0.39f, 1.0f); // Новый серо-голубой фон (#4A5563)**  initGame(); } |
| • Добавлены вызовы функций: glClearColor | |
| **initGame** | |
| void initGame() {  // Инициализация игровой сетки  initGameGrid();   // Инициализация танка игрока  initTank();   // Инициализация пуль  initBullets();   // Инициализация врагов  initEnemies();   // Настраиваем ортографическую проекцию  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glOrtho(-FIELD\_WIDTH / 2, FIELD\_WIDTH / 2 + PANEL\_WIDTH,  -FIELD\_HEIGHT / 2, FIELD\_HEIGHT / 2,  -1.0, 1.0);   // Переключаемся на модель-вид  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();   // Инициализация кнопок меню  **startButton = (Button){ 0.0f, 0.0f, 6.0f, 1.5f, "Start Game" };**  **exitButton = (Button){ 0.0f, -2.0f, 6.0f, 1.5f, "Exit" };**  **okButton = (Button){ -3.0f, -3.0f, 6.0f, 1.5f, "Ok" };**   // Сброс игровых параметров  playerHP = 5;  gameScore = 0;  gameTime = 0;  gameState = GAME\_MENU; } | void initGame() {  // Инициализация игровой сетки  initGameGrid();   // Инициализация танка игрока  initTank();   // Инициализация пуль  initBullets();   // Инициализация врагов  initEnemies();   // Настраиваем ортографическую проекцию  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glOrtho(-FIELD\_WIDTH / 2, FIELD\_WIDTH / 2 + PANEL\_WIDTH,  -FIELD\_HEIGHT / 2, FIELD\_HEIGHT / 2,  -1.0, 1.0);   // Переключаемся на модель-вид  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();   // Инициализация кнопок меню  **startButton = (Button){ 0.0f, 0.0f, 6.0f, 1.5f, "Start Game", -2.0f, 0.0f };**  **exitButton = (Button){ 0.0f, -2.0f, 6.0f, 1.5f, "Exit", -2.0f, 0.0f };**  **okButton = (Button){ -3.0f, -3.0f, 6.0f, 1.5f, "Ok", -2.0f, 0.0f };**   **gameOverButton = (Button){ -4.5f, -3.0f, 4.0f, 1.5f, "Game Over", -1.0f, 0.0f };**  **restartButton = (Button){ 0.5f, -3.0f, 4.0f, 1.5f, "Restart", -1.0f, 0.0f };**    // Сброс игровых параметров  playerHP = 5;  gameScore = 0;  gameTime = 0;  gameState = GAME\_MENU; } |
| • Добавлены присваивания/обновления: startButton = (Button){ 0.0f, 0.0f, 6.0f, 1.5f, "Start Game", -2.0f, 0.0f };; exitButton = (Button){ 0.0f, -2.0f, 6.0f, 1.5f, "Exit", -2.0f, 0.0f };; okButton = (Button){ -3.0f, -3.0f, 6.0f, 1.5f, "Ok", -2.0f, 0.0f };; gameOverButton = (Button){ -4.5f, -3.0f, 4.0f, 1.5f, "Game Over", -1.0f, 0.0f };; restartButton = (Button){ 0.5f, -3.0f, 4.0f, 1.5f, "Restart", -1.0f, 0.0f }; | |
| **initTank** | |
| void initTank() {  playerTank.x = -3.0f; // Стартовая позиция X  playerTank.y = -7.0f; // Стартовая позиция Y  playerTank.direction = 0; // Смотрит вверх  playerTank.isMoving = false;  playerTank.moveProgress = 0.0f;  **// Переводим мировые координаты в клеточные**  **playerTank.targetX = (int)((playerTank.x + 8.5f) / 0.5f);**  **playerTank.targetY = (int)((playerTank.y + 8.5f) / 0.5f);** } | void initTank() {  playerTank.x = -3.0f; // Стартовая позиция X  playerTank.y = -7.0f; // Стартовая позиция Y  playerTank.direction = 0; // Смотрит вверх  playerTank.isMoving = false;  playerTank.moveProgress = 0.0f;  **// Округляем к ближайшему центру сетки (центры лежат на -8.0 + n \* CELL\_SIZE)**  **playerTank.targetX = (int)roundf((playerTank.x + 8.0f) / CELL\_SIZE);**  **playerTank.targetY = (int)roundf((playerTank.y + 8.0f) / CELL\_SIZE);**   **// Жёстко ставим танк в центр вычисленной клетки — так не будет стартового рывка**  **playerTank.x = -8.0f + playerTank.targetX \* CELL\_SIZE;**  **playerTank.y = -8.0f + playerTank.targetY \* CELL\_SIZE;** } |
| • Добавлены вызовы функций: roundf • Добавлены присваивания/обновления: playerTank.targetX = (int)roundf((playerTank.x + 8.0f) / CELL\_SIZE);; playerTank.targetY = (int)roundf((playerTank.y + 8.0f) / CELL\_SIZE);; playerTank.x = -8.0f + playerTank.targetX \* CELL\_SIZE;; playerTank.y = -8.0f + playerTank.targetY \* CELL\_SIZE; | |
| **isAttackLine** | |
| bool isAttackLine(int x, int y) {  **return (y == 2) || (x == 17) || (x == 18);** } | bool isAttackLine(int x, int y) {  **return (y == 1) || (x == 17) || (x == 18);** } |
| • Обновлен код/логика без явных добавлений | |
| **mouse** | |
| void mouse(int button, int state, int x, int y) {  // Преобразование координат мыши в мировые координаты  mouseX = x;  mouseY = y;   if (button == GLUT\_LEFT\_BUTTON) {  mouseLeftDown = (state == GLUT\_DOWN);   if (mouseLeftDown) {  // Преобразуем координаты экрана в мировые  GLint viewport[4];  GLdouble modelview[16], projection[16];  GLfloat winX, winY, winZ;  GLdouble posX, posY, posZ;   glGetIntegerv(GL\_VIEWPORT, viewport);  glGetDoublev(GL\_MODELVIEW\_MATRIX, modelview);  glGetDoublev(GL\_PROJECTION\_MATRIX, projection);   winX = (float)x;  winY = (float)viewport[3] - (float)y;  glReadPixels(x, (int)winY, 1, 1, GL\_DEPTH\_COMPONENT, GL\_FLOAT, &winZ);   gluUnProject(winX, winY, winZ, modelview, projection, viewport, &posX, &posY, &posZ);   float worldX = (float)posX;  float worldY = (float)posY;   // Проверка нажатия на кнопки  if (gameState == GAME\_MENU) {  // Кнопка "Начать игру"  if (worldX >= startButton.x && worldX <= startButton.x + startButton.width &&  worldY >= startButton.y && worldY <= startButton.y + startButton.height) {  gameStartTime = glutGet(GLUT\_ELAPSED\_TIME);  gameState = GAME\_PLAYING;  }  // Кнопка "Выход"  else if (worldX >= exitButton.x && worldX <= exitButton.x + exitButton.width &&  worldY >= exitButton.y && worldY <= exitButton.y + exitButton.height) {  exit(0);  }  }  **// Кнопка "OK" на экранах окончания игры**  **else if (gameState == GAME\_WIN || gameState == GAME\_LOSE) {**  if (worldX >= okButton.x && worldX <= okButton.x + okButton.width &&  worldY >= okButton.y && worldY <= okButton.y + okButton.height) {  **exit(0);**  }  }  }  } } | void mouse(int button, int state, int x, int y) {  // Преобразование координат мыши в мировые координаты  mouseX = x;  mouseY = y;   if (button == GLUT\_LEFT\_BUTTON) {  mouseLeftDown = (state == GLUT\_DOWN);   if (mouseLeftDown) {  // Преобразуем координаты экрана в мировые  GLint viewport[4];  GLdouble modelview[16], projection[16];  GLfloat winX, winY, winZ;  GLdouble posX, posY, posZ;   glGetIntegerv(GL\_VIEWPORT, viewport);  glGetDoublev(GL\_MODELVIEW\_MATRIX, modelview);  glGetDoublev(GL\_PROJECTION\_MATRIX, projection);   winX = (float)x;  winY = (float)viewport[3] - (float)y;  glReadPixels(x, (int)winY, 1, 1, GL\_DEPTH\_COMPONENT, GL\_FLOAT, &winZ);   gluUnProject(winX, winY, winZ, modelview, projection, viewport, &posX, &posY, &posZ);   float worldX = (float)posX;  float worldY = (float)posY;   // Проверка нажатия на кнопки  if (gameState == GAME\_MENU) {  // Кнопка "Начать игру"  if (worldX >= startButton.x && worldX <= startButton.x + startButton.width &&  worldY >= startButton.y && worldY <= startButton.y + startButton.height) {  gameStartTime = glutGet(GLUT\_ELAPSED\_TIME);  gameState = GAME\_PLAYING;  }  // Кнопка "Выход"  else if (worldX >= exitButton.x && worldX <= exitButton.x + exitButton.width &&  worldY >= exitButton.y && worldY <= exitButton.y + exitButton.height) {  exit(0);  }  }   **else if (gameState == GAME\_WIN) {**  **// Нажатие на кнопку OK**  if (worldX >= okButton.x && worldX <= okButton.x + okButton.width &&  worldY >= okButton.y && worldY <= okButton.y + okButton.height) {   **gameState = GAME\_MENU;**  **return;**  }  }   **// Экран проигрыша: две кнопки**  **else if (gameState == GAME\_LOSE) {**  **// перезапуск игры**  **if (worldX >= restartButton.x && worldX <= restartButton.x + restartButton.width &&**  **worldY >= restartButton.y && worldY <= restartButton.y + restartButton.height) {**  **initGame(); // сбрасываем состояние**  **gameState = GAME\_PLAYING; // сразу возвращаемся в игру**  **return;**  **}**  **// GAME OVER**   **if (worldX >= gameOverButton.x && worldX <= gameOverButton.x + gameOverButton.width &&**  **worldY >= gameOverButton.y && worldY <= gameOverButton.y + gameOverButton.height) {**  **gameState = GAME\_MENU; // показываем экран меню**  **return;**  **}**  **}**   }  } } |
| • Добавлены локальные переменные: return • Добавлены вызовы функций: initGame • Добавлены условия: else if (gameState == GAME\_WIN) {; else if (gameState == GAME\_LOSE) {; if (worldX >= restartButton.x && worldX <= restartButton.x + restartButton.width &&; if (worldX >= gameOverButton.x && worldX <= gameOverButton.x + gameOverButton.width && • Добавлены присваивания/обновления: gameState = GAME\_MENU;; gameState = GAME\_PLAYING;; gameState = GAME\_MENU; | |
| **respawnEnemies** | |
| void respawnEnemies() {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);  int gameDuration = (currentTime - gameStartTime) / 1000;   enemiesSpawnCount = 1 + gameDuration / 60;  if (enemiesSpawnCount > 5) enemiesSpawnCount = 5;   if (currentTime - lastRespawnTime < respawnDelay) return;   int spawned = 0;  for (int i = 0; i < MAX\_ENEMIES && spawned < enemiesSpawnCount; i++) {  if (!enemies[i].active) {  int spawnIndex = rand() % 5;  enemies[i].x = spawnPoints[spawnIndex][0];  enemies[i].y = spawnPoints[spawnIndex][1];  enemies[i].direction = 2;  enemies[i].active = true;  enemies[i].lastShotTime = currentTime;   // Инициализация параметров движения  enemies[i].isMoving = false;  enemies[i].moveProgress = 0.0f;  **enemies[i].targetX = (int)((enemies[i].x + 8.5f) / 0.5f);**  **enemies[i].targetY = (int)((enemies[i].y + 8.5f) / 0.5f);**  enemies[i].path = NULL;  enemies[i].pathStep = 0;  enemies[i].pathLength = 0;  // Случайная задержка между движениями (500-1000 мс)  enemies[i].moveDelay = 500 + rand() % 500;  enemies[i].lastMoveTime = currentTime;   // Находим путь к линии атаки  findPathToAttackLine(&enemies[i]);   spawned++;  }  }   lastRespawnTime = currentTime; } | void respawnEnemies() {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);  int gameDuration = (currentTime - gameStartTime) / 1000;   enemiesSpawnCount = 1 + gameDuration / 60;  if (enemiesSpawnCount > 5) enemiesSpawnCount = 5;   if (currentTime - lastRespawnTime < respawnDelay) return;   int spawned = 0;  for (int i = 0; i < MAX\_ENEMIES && spawned < enemiesSpawnCount; i++) {  if (!enemies[i].active) {  int spawnIndex = rand() % 5;  enemies[i].x = spawnPoints[spawnIndex][0];  enemies[i].y = spawnPoints[spawnIndex][1];  enemies[i].direction = 2;  enemies[i].active = true;  enemies[i].lastShotTime = currentTime;   // Инициализация параметров движения  enemies[i].isMoving = false;  enemies[i].moveProgress = 0.0f;  **enemies[i].targetX = (int)roundf((enemies[i].x + 8.0f) / 0.5f);**  **enemies[i].targetY = (int)roundf((enemies[i].y + 8.0f) / 0.5f);**  enemies[i].path = NULL;  enemies[i].pathStep = 0;  enemies[i].pathLength = 0;  // Случайная задержка между движениями (500-1000 мс)  enemies[i].moveDelay = 500 + rand() % 500;  enemies[i].lastMoveTime = currentTime;   // Находим путь к линии атаки  findPathToAttackLine(&enemies[i]);   spawned++;  }  }   lastRespawnTime = currentTime; } |
| • Добавлены вызовы функций: roundf • Добавлены присваивания/обновления: enemies[i].targetX = (int)roundf((enemies[i].x + 8.0f) / 0.5f);; enemies[i].targetY = (int)roundf((enemies[i].y + 8.0f) / 0.5f); | |
| **specialKeys** | |
| void specialKeys(int key, int x, int y) {  int newDirection = -1;   switch (key) {  case GLUT\_KEY\_UP: newDirection = 0; break;  case GLUT\_KEY\_LEFT: newDirection = 1; break;  case GLUT\_KEY\_DOWN: newDirection = 2; break;  case GLUT\_KEY\_RIGHT: newDirection = 3; break;  }   if (newDirection != -1 && !playerTank.isMoving) {  // Если нужно повернуться  if (playerTank.direction != newDirection) {  playerTank.direction = newDirection;  }  // Если можно двигаться  else {  int nextX = playerTank.targetX;  int nextY = playerTank.targetY;   switch (newDirection) {  **case 0: nextY += 2; break; // Вверх**  **case 1: nextX -= 2; break; // Влево**  **case 2: nextY -= 2; break; // Вниз**  **case 3: nextX += 2; break; // Вправо**  }   if (canMoveTo(nextX, nextY)) {  playerTank.targetX = nextX;  playerTank.targetY = nextY;  playerTank.isMoving = true;  playerTank.moveProgress = 0.0f;  }  }  }  glutPostRedisplay(); } | void specialKeys(int key, int x, int y) {  int newDirection = -1;   switch (key) {  case GLUT\_KEY\_UP: newDirection = 0; break;  case GLUT\_KEY\_LEFT: newDirection = 1; break;  case GLUT\_KEY\_DOWN: newDirection = 2; break;  case GLUT\_KEY\_RIGHT: newDirection = 3; break;  }   if (newDirection != -1 && !playerTank.isMoving) {  // Если нужно повернуться  if (playerTank.direction != newDirection) {  playerTank.direction = newDirection;  }  // Если можно двигаться  else {  int nextX = playerTank.targetX;  int nextY = playerTank.targetY;   switch (newDirection) {  **case 0: nextY += 1; break; // Вверх**  **case 1: nextX -= 1; break; // Влево**  **case 2: nextY -= 1; break; // Вниз**  **case 3: nextX += 1; break; // Вправо**  }   if (canMoveTo(nextX, nextY)) {  playerTank.targetX = nextX;  playerTank.targetY = nextY;  playerTank.isMoving = true;  playerTank.moveProgress = 0.0f;  }  }  }  glutPostRedisplay(); } |
| • Добавлены присваивания/обновления: case 0: nextY += 1; break;; case 1: nextX -= 1; break;; case 2: nextY -= 1; break;; case 3: nextX += 1; break; | |
| **updateBullets** | |
| void updateBullets() {  **const float BULLET\_SPEED = 1.0f;**   for (int i = 0; i < MAX\_BULLETS; i++) {  if (!bullets[i].active) continue;   // Проверка попадания в игрока (только для вражеских пуль)  if (bullets[i].isEnemy) {  float distX = fabsf(bullets[i].x - playerTank.x);  float distY = fabsf(bullets[i].y - playerTank.y);   if (distX < 0.5f && distY < 0.5f) {  bullets[i].active = false;  // Обработка попадания в игрока  playerHP--;   // Проверка проигрыша по здоровью  if (playerHP <= 0) {  gameState = GAME\_LOSE;  }  }  }  // Проверка попадания во врагов (только для пуль игрока)  if (!bullets[i].isEnemy) {  for (int j = 0; j < MAX\_ENEMIES; j++) {  if (enemies[j].active) {  // Проверка столкновения с учетом размеров  float dx = fabsf(bullets[i].x - enemies[j].x);  float dy = fabsf(bullets[i].y - enemies[j].y);   // Размер танка: 1.0x1.0, размер пули: 0.3x0.3  if (dx < 0.65f && dy < 0.65f) {  bullets[i].active = false;  enemies[j].active = false;   // Начисление 50 очков за врага  gameScore += 50;   // Уменьшаем время респавна  respawnDelay -= 500;  if (respawnDelay < 1000) respawnDelay = 1000;  break;  }  }  }  }   // Двигаем пулю в зависимости от направления  switch (bullets[i].direction) {  case 0: bullets[i].y += BULLET\_SPEED \* 0.05f; break;  case 1: bullets[i].x -= BULLET\_SPEED \* 0.05f; break;  case 2: bullets[i].y -= BULLET\_SPEED \* 0.05f; break;  case 3: bullets[i].x += BULLET\_SPEED \* 0.05f; break;  }   // Проверка выхода за границы поля  if (bullets[i].x < -9.0f || bullets[i].x > 9.0f ||  bullets[i].y < -9.0f || bullets[i].y > 9.0f) {  bullets[i].active = false;  continue;  }   // Проверка столкновений  int gridX = (int)((bullets[i].x + 8.5f) / 0.5f);  int gridY = (int)((bullets[i].y + 8.5f) / 0.5f);    if (gridX >= 0 && gridX < GRID\_WIDTH && gridY >= 0 && gridY < GRID\_HEIGHT) {  GameObject\* obj = &gameGrid[gridY][gridX];   // Уничтожение базы  if (obj->type == OBJECT\_BASE) {  gameState = GAME\_LOSE;  }   if (!obj->passable) {  // Начисляем очки за разрушение кирпича  if ((obj->type == OBJECT\_BRICK || obj->type == OBJECT\_BRICK\_BASE) && obj->durability > 0) {  gameScore += 10;  }   // Для разрушаемых объектов уменьшаем прочность  if (obj->destructible && obj->durability > 0) {  obj->durability--;   // Если прочность закончилась - удаляем объект  if (obj->durability <= 0) {  obj->type = OBJECT\_EMPTY;  obj->passable = true;  }  }   bullets[i].active = false;  }  }  } } | void updateBullets() {  **const float BULLET\_SPEED = 0.4f;**   for (int i = 0; i < MAX\_BULLETS; i++) {  if (!bullets[i].active) continue;   // Проверка попадания в игрока (только для вражеских пуль)  if (bullets[i].isEnemy) {  float distX = fabsf(bullets[i].x - playerTank.x);  float distY = fabsf(bullets[i].y - playerTank.y);   if (distX < 0.5f && distY < 0.5f) {  bullets[i].active = false;  // Обработка попадания в игрока  playerHP--;   // Проверка проигрыша по здоровью  if (playerHP <= 0) {  gameState = GAME\_LOSE;  }  }  }  // Проверка попадания во врагов (только для пуль игрока)  if (!bullets[i].isEnemy) {  for (int j = 0; j < MAX\_ENEMIES; j++) {  if (enemies[j].active) {  // Проверка столкновения с учетом размеров  float dx = fabsf(bullets[i].x - enemies[j].x);  float dy = fabsf(bullets[i].y - enemies[j].y);   // Размер танка: 1.0x1.0, размер пули: 0.3x0.3  if (dx < 0.65f && dy < 0.65f) {  bullets[i].active = false;  enemies[j].active = false;   // Начисление 50 очков за врага  gameScore += 50;   // Уменьшаем время респавна  respawnDelay -= 500;  if (respawnDelay < 1000) respawnDelay = 1000;  break;  }  }  }  }   // Двигаем пулю в зависимости от направления  switch (bullets[i].direction) {  case 0: bullets[i].y += BULLET\_SPEED \* 0.05f; break;  case 1: bullets[i].x -= BULLET\_SPEED \* 0.05f; break;  case 2: bullets[i].y -= BULLET\_SPEED \* 0.05f; break;  case 3: bullets[i].x += BULLET\_SPEED \* 0.05f; break;  }   // Проверка выхода за границы поля  if (bullets[i].x < -9.0f || bullets[i].x > 9.0f ||  bullets[i].y < -9.0f || bullets[i].y > 9.0f) {  bullets[i].active = false;  continue;  }   // Проверка столкновений  int gridX = (int)((bullets[i].x + 8.5f) / 0.5f);  int gridY = (int)((bullets[i].y + 8.5f) / 0.5f);    if (gridX >= 0 && gridX < GRID\_WIDTH && gridY >= 0 && gridY < GRID\_HEIGHT) {  GameObject\* obj = &gameGrid[gridY][gridX];   // Уничтожение базы  if (obj->type == OBJECT\_BASE) {  gameState = GAME\_LOSE;  }   if (!obj->passable) {  // Начисляем очки за разрушение кирпича  if ((obj->type == OBJECT\_BRICK || obj->type == OBJECT\_BRICK\_BASE) && obj->durability > 0) {  gameScore += 10;  }   // Для разрушаемых объектов уменьшаем прочность  if (obj->destructible && obj->durability > 0) {  obj->durability--;   // Если прочность закончилась - удаляем объект  if (obj->durability <= 0) {  obj->type = OBJECT\_EMPTY;  obj->passable = true;  }  }   bullets[i].active = false;  }  }  } } |
| • Добавлены присваивания/обновления: const float BULLET\_SPEED = 0.4f; | |
| **updateEnemies** | |
| void updateEnemies() {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);   for (int i = 0; i < MAX\_ENEMIES; i++) {  if (!enemies[i].active) continue;   **// Текущая позиция в сетке**  **int currentGridX = (int)((enemies[i].x + 8.5f) / 0.5f);**  **int currentGridY = (int)((enemies[i].y + 8.5f) / 0.5f);**   **// Определяем состояние (на линии атаки или нет)**  enemies[i].state = isAttackLine(currentGridX, currentGridY) ? 1 : 0;   **// Стрельба врагов (раз в 2 секунды)**  if (currentTime - enemies[i].lastShotTime > 2000) {  createEnemyBullet(i);  enemies[i].lastShotTime = currentTime;  }   **// Если танк движется**  if (enemies[i].isMoving) {  enemies[i].moveProgress += 0.03f;   if (enemies[i].moveProgress >= 1.0f) {  **// Завершение движения**  enemies[i].isMoving = false;  enemies[i].moveProgress = 1.0f;  enemies[i].lastMoveTime = currentTime;   **// Финализируем позицию**  enemies[i].x = -8.5f + enemies[i].targetX \* 0.5f + 0.5f;  enemies[i].y = -8.5f + enemies[i].targetY \* 0.5f + 0.5f;   **// Обновляем текущую позицию в сетке**  currentGridX = enemies[i].targetX;  currentGridY = enemies[i].targetY;   **// Если достигли точки пути, переходим к следующей**  if (enemies[i].path != NULL && enemies[i].pathStep < enemies[i].pathLength - 1) {  enemies[i].pathStep++;  **enemies[i].targetX = enemies[i].path[enemies[i].pathStep \* 2];**  **enemies[i].targetY = enemies[i].path[enemies[i].pathStep \* 2 + 1];**  }  }  else {  **// Плавное перемещение к цели**  float targetWorldX = -8.5f + enemies[i].targetX \* 0.5f + 0.5f;  float targetWorldY = -8.5f + enemies[i].targetY \* 0.5f + 0.5f;   **enemies[i].x += (targetWorldX - enemies[i].x) \* 0.1f;**  **enemies[i].y += (targetWorldY - enemies[i].y) \* 0.1f;**  }  }  **// Если не движется и пришло время двигаться**  else if (currentTime - enemies[i].lastMoveTime > enemies[i].moveDelay) {  **int newDirection = -1;**  **int nextGridX = currentGridX;**  **int nextGridY = currentGridY;**   **// Определяем следующее движение**  **if (enemies[i].path != NULL && enemies[i].pathStep < enemies[i].pathLength) {**  **nextGridX = enemies[i].path[enemies[i].pathStep \* 2];**  **nextGridY = enemies[i].path[enemies[i].pathStep \* 2 + 1];**   **// Вычисляем правильное направление**  **if (nextGridX > currentGridX) newDirection = 3; // Вправо**  **else if (nextGridX < currentGridX) newDirection = 1; // Влево**  **else if (nextGridY > currentGridY) newDirection = 0; // Вверх**  **else newDirection = 2; // Вниз**  **}**  **// Если на линии атаки (состояние 1)**  **else if (enemies[i].state == 1) {**  **if (currentGridX == 17 || currentGridX == 18) {**  **// На вертикальной линии - двигаемся вниз**  **newDirection = 2;**  **nextGridY = currentGridY - 2;**  **}**  **else if (currentGridY == 1) {**  **// На горизонтальной линии - двигаемся к центру**  **if (currentGridX < 17) {**  **newDirection = 3; // Вправо**  **nextGridX = currentGridX + 2;**  **}**  **else {**  **newDirection = 1; // Влево**  **nextGridX = currentGridX - 2;**  **}**  }  }   **// Если направление определено и движение возможно**  **if (newDirection != -1 && canMoveTo(nextGridX, nextGridY)) {**  **enemies[i].direction = newDirection;**  **enemies[i].targetX = nextGridX;**  **enemies[i].targetY = nextGridY;**  **enemies[i].isMoving = true;**  **enemies[i].moveProgress = 0.0f;**  **}**  **// Если не можем двигаться, ждем**  **else {**  **enemies[i].lastMoveTime = currentTime;**  }  }   **// Корректировка направления для танков на линии атаки**  **if (enemies[i].state == 1) {**  **if (currentGridY == 2) {**  **// На горизонтальной линии - поворачиваем к базе**  **enemies[i].direction = (currentGridX < 17) ? 3 : 1; // Вправо/Влево**  }  else if (currentGridX == 17 || currentGridX == 18) {  **// На вертикальной линии - поворачиваем вниз**  **enemies[i].direction = 2;**  }  }  } } | void updateEnemies() {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);   for (int i = 0; i < MAX\_ENEMIES; i++) {  if (!enemies[i].active) continue;   **// Текущая клетка врага**  **int currentGridX = (int)roundf((enemies[i].x + 8.0f) / 0.5f);**  **int currentGridY = (int)roundf((enemies[i].y + 8.0f) / 0.5f);**   **// на линии атаки или нет**  enemies[i].state = isAttackLine(currentGridX, currentGridY) ? 1 : 0;   **// Стрельба раз в 2 секунды**  if (currentTime - enemies[i].lastShotTime > 2000) {  createEnemyBullet(i);  enemies[i].lastShotTime = currentTime;  }   **// Движемся к целевой клетке**  if (enemies[i].isMoving) {  enemies[i].moveProgress += 0.03f;   **// Завершили шаг**  if (enemies[i].moveProgress >= 1.0f) {  enemies[i].isMoving = false;  enemies[i].moveProgress = 1.0f;  enemies[i].lastMoveTime = currentTime;   **// Фиксируем центр целевой клетки**  enemies[i].x = -8.5f + enemies[i].targetX \* 0.5f + 0.5f;  enemies[i].y = -8.5f + enemies[i].targetY \* 0.5f + 0.5f;   **// Обновляем текущую клетку**  currentGridX = enemies[i].targetX;  currentGridY = enemies[i].targetY;   **// Переходим к следующей точке пути**  if (enemies[i].path != NULL && enemies[i].pathStep < enemies[i].pathLength - 1) {  enemies[i].pathStep++;  **}**  **// Путь завершён — очищаем**  **else if (enemies[i].path != NULL) {**  **free(enemies[i].path);**  **enemies[i].path = NULL;**  **enemies[i].pathLength = 0;**  }  }  **// В пути: выставляем направление по вектору на цель**  else {  float targetWorldX = -8.5f + enemies[i].targetX \* 0.5f + 0.5f;  float targetWorldY = -8.5f + enemies[i].targetY \* 0.5f + 0.5f;   **float dx = targetWorldX - enemies[i].x;**  **float dy = targetWorldY - enemies[i].y;**   **if (fabsf(dx) > fabsf(dy)) {**  **enemies[i].direction = (dx > 0.0f) ? 3 : 1; // вправо : влево**  **}**  **else {**  **enemies[i].direction = (dy > 0.0f) ? 0 : 2; // вверх : вниз**  **}**  }  }  **// Стоим: решаем, куда сделать следующий шаг**  else if (currentTime - enemies[i].lastMoveTime > enemies[i].moveDelay) {  **// A) Есть путь — идём по нему**  **if (enemies[i].path != NULL && enemies[i].pathStep < enemies[i].pathLength) {**  **int nextGridX = enemies[i].path[enemies[i].pathStep \* 2];**  **int nextGridY = enemies[i].path[enemies[i].pathStep \* 2 + 1];**   **if (canMoveTo(nextGridX, nextGridY)) {**  **enemies[i].targetX = nextGridX;**  **enemies[i].targetY = nextGridY;**  **enemies[i].isMoving = true;**  **enemies[i].moveProgress = 0.0f;**  }  }  **// B) Путь пуст и мы на линии атаки**   **else if (enemies[i].path == NULL && enemies[i].state == 1) {**  **int newDirection = -1;**  **int nextGridX = currentGridX;**  **int nextGridY = currentGridY;**   **// Горизонтальная нижняя линия**  **if (currentGridY == 1) {**  **if (currentGridX < 17) { newDirection = 3; nextGridX++; } // вправо**  **else if (currentGridX > 18) { newDirection = 1; nextGridX--; } // влево**  **}**  **// Вертикальные направляющие к базе**  **else if (currentGridX == 17 || currentGridX == 18) {**  **if (currentGridY > 1) { newDirection = 2; nextGridY--; } // вниз**  **}**   **if (newDirection != -1 && canMoveTo(nextGridX, nextGridY)) {**  **enemies[i].targetX = nextGridX;**  **enemies[i].targetY = nextGridY;**  **enemies[i].isMoving = true;**  **enemies[i].moveProgress = 0.0f;**  **}**  }  }   **// Стоим на линии атаки без пути — смотрим в нужную сторону**  **if (!enemies[i].isMoving && enemies[i].path == NULL && enemies[i].state == 1) {**  **if (currentGridY == 1) {**  **if (currentGridX < 17) enemies[i].direction = 3; // вправо**  **else if (currentGridX > 18) enemies[i].direction = 1; // влево**  **else enemies[i].direction = 2; // вниз**  }  else if (currentGridX == 17 || currentGridX == 18) {  **enemies[i].direction = 2; // вниз**  }  }  } } |
| • Добавлены вызовы функций: canMoveTo, fabsf, free, roundf • Добавлены условия: else if (enemies[i].path != NULL) {; if (fabsf(dx) > fabsf(dy)) {; if (enemies[i].path != NULL && enemies[i].pathStep < enemies[i].pathLength) {; if (canMoveTo(nextGridX, nextGridY)) {; else if (enemies[i].path == NULL && enemies[i].state == 1) {; if (currentGridY == 1) {; if (currentGridX < 17) { newDirection = 3; nextGridX++; }; else if (currentGridX > 18) { newDirection = 1; nextGridX--; }; else if (currentGridX == 17 || currentGridX == 18) {; if (currentGridY > 1) { newDirection = 2; nextGridY--; }; if (newDirection != -1 && canMoveTo(nextGridX, nextGridY)) {; if (!enemies[i].isMoving && enemies[i].path == NULL && enemies[i].state == 1) {; if (currentGridY == 1) {; if (currentGridX < 17) enemies[i].direction = 3;; else if (currentGridX > 18) enemies[i].direction = 1; • Добавлены присваивания/обновления: int currentGridX = (int)roundf((enemies[i].x + 8.0f) / 0.5f);; int currentGridY = (int)roundf((enemies[i].y + 8.0f) / 0.5f);; enemies[i].path = NULL;; enemies[i].pathLength = 0;; float dx = targetWorldX - enemies[i].x;; float dy = targetWorldY - enemies[i].y;; enemies[i].direction = (dx > 0.0f) ? 3 : 1;; enemies[i].direction = (dy > 0.0f) ? 0 : 2;; int nextGridX = enemies[i].path[enemies[i].pathStep \* 2];; int nextGridY = enemies[i].path[enemies[i].pathStep \* 2 + 1];; enemies[i].targetX = nextGridX;; enemies[i].targetY = nextGridY;; enemies[i].isMoving = true;; enemies[i].moveProgress = 0.0f;; int newDirection = -1;; int nextGridX = currentGridX;; int nextGridY = currentGridY;; if (currentGridX < 17) { newDirection = 3; nextGridX++; }; else if (currentGridX > 18) { newDirection = 1; nextGridX--; }; if (currentGridY > 1) { newDirection = 2; nextGridY--; }; enemies[i].targetX = nextGridX;; enemies[i].targetY = nextGridY;; enemies[i].isMoving = true;; enemies[i].moveProgress = 0.0f;; if (currentGridX < 17) enemies[i].direction = 3;; else if (currentGridX > 18) enemies[i].direction = 1;; else enemies[i].direction = 2;; enemies[i].direction = 2; | |
| **updateGameTime** | |
| void updateGameTime() {  if (gameState == GAME\_PLAYING) {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);  gameTime = (currentTime - gameStartTime) / 1000;   // Проверка победы  **if (gameScore >= 10000) {**  gameState = GAME\_WIN;  }  } } | void updateGameTime() {  if (gameState == GAME\_PLAYING) {  int currentTime = glutGet(GLUT\_ELAPSED\_TIME);  gameTime = (currentTime - gameStartTime) / 1000;   // Проверка победы  **if (gameScore >= 2000) {**  gameState = GAME\_WIN;  }  } } |
| • Добавлены условия: if (gameScore >= 2000) { | |
| **updateTank** | |
| void updateTank() {  if (playerTank.isMoving) {  **playerTank.moveProgress += 0.05f; // Скорость движения**   if (playerTank.moveProgress >= 1.0f) {  playerTank.isMoving = false;  playerTank.moveProgress = 1.0f;  }   // Рассчет новой позиции  float targetWorldX = -8.5f + playerTank.targetX \* 0.5f + 0.5f;  float targetWorldY = -8.5f + playerTank.targetY \* 0.5f + 0.5f;   playerTank.x = playerTank.x + (targetWorldX - playerTank.x) \* playerTank.moveProgress;  playerTank.y = playerTank.y + (targetWorldY - playerTank.y) \* playerTank.moveProgress;  } } | void updateTank() {  if (playerTank.isMoving) {  **playerTank.moveProgress += 0.020f; // Скорость движения**   if (playerTank.moveProgress >= 1.0f) {  playerTank.isMoving = false;  playerTank.moveProgress = 1.0f;  }   // Рассчет новой позиции  float targetWorldX = -8.5f + playerTank.targetX \* 0.5f + 0.5f;  float targetWorldY = -8.5f + playerTank.targetY \* 0.5f + 0.5f;   playerTank.x = playerTank.x + (targetWorldX - playerTank.x) \* playerTank.moveProgress;  playerTank.y = playerTank.y + (targetWorldY - playerTank.y) \* playerTank.moveProgress;  } } |
| • Добавлены присваивания/обновления: playerTank.moveProgress += 0.020f; | |