ДЕМОНСТРАЦИЯ ВОЗМОЖНОСТЕЙ ДВИЖКА UNITY3D НА ПРИМЕРЕ КОМПЬЮТЕРНОЙ ИГРЫ STARKOMBAT

ст.гр.ПОВТ-1-10 Иванский С.В., н.рук. доц. Мусина И.Р.
КГТУ им. И. Раззакова

Приводятся результаты разработки компьютерной игры космической тематики StarKombat на основе применения движка Unity3D, который позволяет быстро и эффективно создавать компьютерные игры любого жанра.

Unity3D – очень популярный движок для создания компьютерных игр. Игровой движок Unity3D называют мультиплатформенной программой для разработки 2D и 3D игровых проектов. В Unity3D имеется свой встроенный генератор ландшафтов и встроенная поддержка сети. Распространяется движок Unity по различным лицензиям, среди которых есть и бесплатная версия Unity Free. Созданные на этом движке игры работают не только на персональных компьютерах, но также на таких приставках, как Xbox и Sony PlayStation3. Игры и программы, разработанные на движке Unity3D, поддерживают такие библиотеки, как DirectX и OpenGL. Движок Unity3D поддерживает известные языки программирования, в частности C# и JavaScript, поэтому в нём можно делать не только игры, но и даже компьютерные программы. Игровой движок Unity3D имеет возможности, которые отличают его от других программ для создания компьютерных игр: высокие возможности импорта данных, мощная гибкость движка, кроссплатформенность, удобный и лёгкий в усвоении пользовательский интерфейс, поддержка физики Physics, создание игр в режиме Real-time.

На рис.1. представлена среда разработки **Unity3D,** которая содержит все компоненты, из которых будет состоять наша игра: рисунки, трёхмерные модели, звуки и музыку, файлы с кодом и игровые сцены. **Игровая сцена** – это уровень игры, отдельное игровое пространство, где всё перечисленное выше соединяется в единое целое – в готовую игру.

Рис.1. Среда разработки Unity3D

Целю работы является разработка компьютерной игры StarKombat с использованием языка JavaScript, которая позволила бы исследовать возможности движка Unity3D.

«StarKombat» – это компьютерная игра, в которой участники управляют звездолётом. На рис. 2. представлена 1 из 10 моделей звездолётов под названием Hunter. Все 10 моделей звездолётов представлены на рис. 3. Суть игры заключается в том, что управляя игровой моделью звездолёта нужно обстреливать противника и при этом уходить от его атак. Побеждает тот, кто первым разрушит звездолёт противника, т.е. количество хит пойнтов «HP» или «Life» (степень целости звездолёта) достигнет нуля. Игра StarKombat требует высокой скорости реакции, тактики, сообразительности (т.е. нужно делать выбор между атакой и защитой, контролировать расход патронов, атаковать с таким расчётом, чтобы попасть в постоянно двигающегося соперника), поэтому данная игра должна способствовать повышению интеллектуальных навыков у играющего.

Рис.2. Модель звездолёта «Hunter»

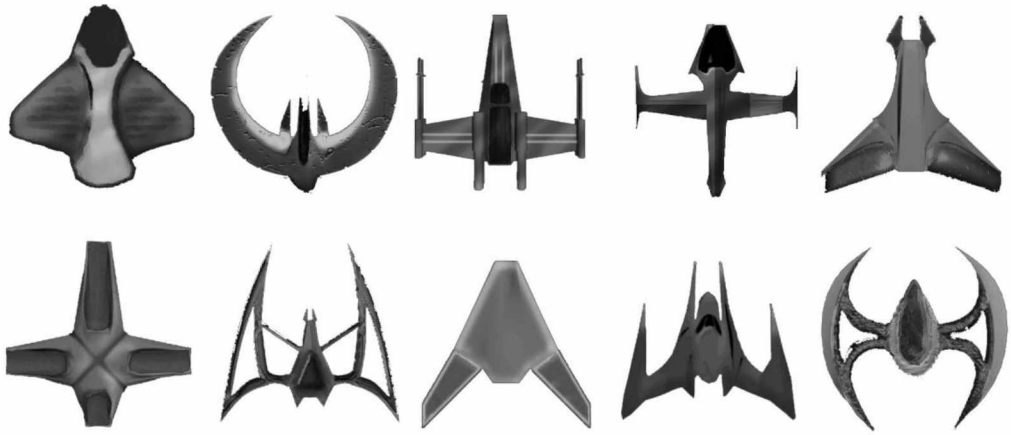

Рис.3. Десять моделей звездолётов

На сегодняшний день существует множество различных аналогов данной игры, таких как *«Звёздный защитник», «Удар с космоса»* и др. В них игроку предлагается уничтожать движущиеся навстречу объекты, постепенно продвигаясь к финишу. Движения объектов осуществляются по заданному алгоритму, поэтому становятся предсказуемыми. Разрабатываемая игра StarKombat отличается тем, что игрокам предстоит биться друг с другом. Здесь нет уровней и количества заработанных очков, игроки борются за первенство друг с другом, тренируя свои игровые навыки.

Были разработаны следующие принципы игры:

- У звездолёта должно быть заданное количество патронов и хит пойнтов «HP» (если оно достигнет нуля, звездолёт будет уничтожен, и игрок проиграет).
- Должно быть соблюдение баланса игры, который заключается в том, что игроки должны находиться в равных условиях. Преимущество в вооружении и разный уровень HP не гарантирует победу. Победа в большей степени должна зависеть от навыков самих участников. В начале игры участники имеют равное количество HP и патронов. Также соблюдается баланс для оружия и защиты (например, более мощное оружие имеет меньшую скорость выстрелов или меньшее количество патронов).
- Ограничения:

  a) игровая карта имеет границы, игроки не могут перемещаться дальше пределов своей зоны, не могут сталкиваться друг с другом;

  b) передвигаться можно лишь в одной плоскости, т.е. нельзя летать вверх и вниз;

  c) отсутствуют повороты, т.е. стрелять можно только вперёд;

  d) инвентарь ограничен 4-мя слотами (если все они будут заполнены, то новые предметы не смогут захватываться).

Также я сформулировал следующие требования к разрабатываемой игре:

- Должен быть выбор режима отображения (оконный или полноэкранный), а также выбор разрешения экрана.
- Должны быть предусмотрены режимы для одиночной игры и игры для двух игроков за одним компьютером.
- Управление моделями звездолётов должно осуществляться посредством клавиатуры. Для навигации в главном меню и меню настроек должна быть предусмотрена мышь.
- Возможности игроков:

  a) передвижения звездолётов (влево, вправо, вперёд и назад);

  b) ускорение (возможность передвигаться быстрее);

  c) уклонение от атак;

  d) атака, нанесение повреждений;

  e) временная защита от повреждений;

  f) разрушение пролетающих мимо объектов;

  g) захват выпадающих из разрушенных объектов бонусов;

  h) хранение полученных предметов в слотах инвентаря, выбор и активация их, либо выброс ненужных.
- Должны быть предусмотрены настройки графики, звука, управления и уровня сложности.

В моей игре игровое пространство представлено видом сверху. В верхней и нижней части экрана находятся звездолёты игроков (см. рис.4.). Для каждого игрока есть своя информационная панель (см. рис. 5.), на которой отображаются уровень HP (life), вид оружия, количество патронов, вид защиты и слоты для предметов.

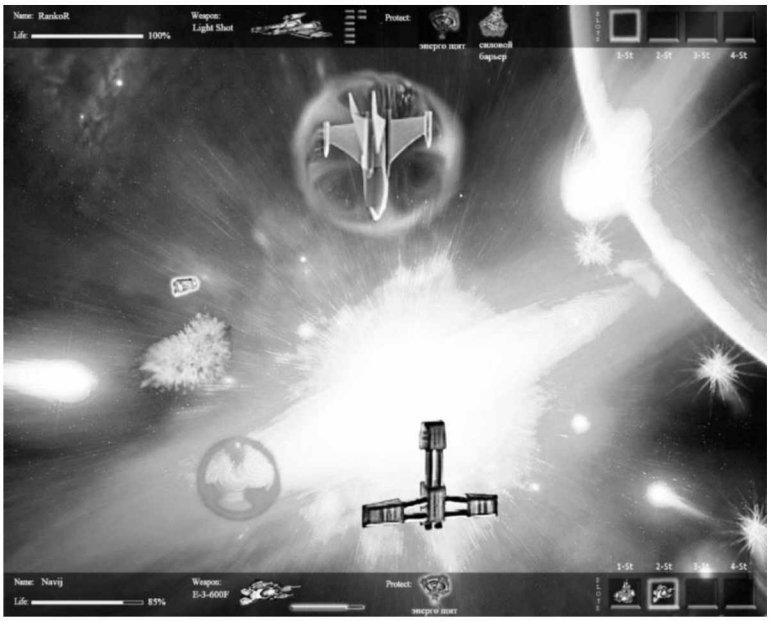

Рис. 4. Экран игры StarKombat

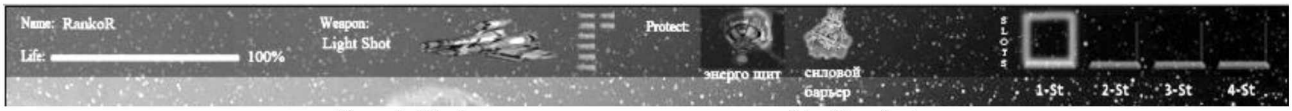

Рис. 5. Информационная панель в игре StarKombat

В сцене также присутствует камера, которая визуализирует нашу игру. Камера может перемещаться в пространстве. Существует проблема, которая заставляет выбирать между свободой перемещения игроков и видимостью их на экране. Когда звездолёты уходят за пределы экрана, мы не можем видеть их. Чтобы оба звездолёта всегда были нам видны, камера должна отодвигаться дальше, если один из звездолётов отходит далеко от центра игрового поля. Это позволяет значительно увеличить пространство перемещения. Но тут возникает другая проблема, – если камера будет отодвигаться слишком далеко, звездолёты станут настолько маленькими (будут настолько далеки от нас), что мы опять не сможем увидеть их. Поэтому это вынуждает ограничивать перемещения. При достижении определённого расстояния звездолёты не могут двигаться дальше. Но благодаря динамическому перемещению камеры это расстояние удалось увеличить в несколько раз.

Управление звездолётом включает перемещение в пространстве, стрельбу и защиту. Клавиши управления я определил в классах Player1 и Player2, которые будут брать данные из файлов. А все скрипты *.js, использующие клавиатурное управление будут брать данные из этих классов. Это позволяет мне изменять кнопки управления только в этих двух классах, а не в каждом файле.

Для перемещения используется встроенный компонент Unity3D – ***ConstantForce***, который имитирует инерцию движения. Таким образом, при нажатии клавиш влево, вправо, вверх, вниз изменяются не координаты звездолётов (x,y), а инерция, которая толкает их в нужную нам сторону. Это обеспечивает плавность движений. При движении звездолёта влево и вправо для большей реалистичности меняется его угол наклона.

Определённому **виду оружия** задаётся свой **тип снаряда**, а также следующие параметры:

1. максимально возможное количество патронов,
2. текущее количество патронов,
3. время задержки между выстрелами.

Снаряду назначаются следующие свойства:

1. скорость полёта,
2. взрыв по времени или взрыв при столкновении,
3. сила урона при столкновении со снарядом,
4. сила урона от взрыва (радиус силы),
5. клонирование снарядов.

Для снарядов создаётся специальный **префаб**, который хранит копию снаряда. Префабы удобно использовать для множественного клонирования.

Когда игрок начинает стрелять, выбранное им оружие создаёт копию своего снаряда. Координаты оружия совпадают с координатами стреляющего игрока, и появившийся снаряд оказывается на том же месте. Свойство скорости полёта заставляет снаряд двигаться вперёд.

Для того, чтобы реализовать **столкновение** пули со звездолётом противника, нужно

определить такие координаты, при совпадении которых произойдёт вызов функции **OnCollision**(). В Unity3D, как и в любом другом движке, есть встроенный компонент **Collider**. Этот компонент автоматически строит координаты столкновения, располагая их в определённой геометрической фигуре, а также детектирует любые соприкосновения этих фигур. Заданный звездолёту детектор столкновений ***Collider -> Mesh*** будет повторять форму звездолёта. Пуля имеет более простую форму, поэтому ей можно назначить и параллелепипед. При соприкосновении рёбер колайдеров пули и звездолёта произойдёт событие столкновения и вызов функции **OnCollisionEnter**(), в которой реализовано всё, что должно произойти при этом столкновении, а именно:

1. исчезновение пули с возникновением искры,
2. отнятие **HP** звездолёта, равное **damage** пули.

Но тут возникает следующая проблема: так как у каждого звездолёта есть свой collider, и пуля имеет свой собственный collider, а столкновение Collision определяется с любым первым встречным колайдером, то при стрельбе пуля столкнётся с выстрелившем её. То есть получается, что игрок будет стрелять сам в себя. Чтобы решить эту проблему я поместил каждый звездолёт на отдельный слой (**Layer**), а в свойствах проекта отключил Collision для совпадающих слоёв. Это значит, что пуля Layer1 не столкнётся со звездолётом Layer1, а столкнётся с любым другим объектом, если его слой не Layer1, в том числе и со звездолётом противника, у которого Layer2.

Но префаб пули 1 на всех, и при клонировании этого объекта его слой остаётся таким, каким он был задан в первоначальном префабе. Я не могу назначить пуле Layer1 или Layer2, иначе 1 игрок будет стрелять нормально, а другой сам в себя. Поэтому необходимо предусмотреть автоматическое определение слоя, который будет совпадать со слоем игрока, что я и сделал.

Определённому **виду защиты** задаётся свой **тип щита**, а также следующие параметры:

1. время действия,
2. уровень HP,
3. движения для подвижных частей щита.

Для щита создаётся специальный префаб, который хранит копию щита (см. рис. 6.). Когда игрок активирует защиту, пред его звездолётом появляется щит, который блокирует пули. Щит использует скрипт распределения слоёв, чтобы при выстрелах пули не сталкивались с щитом этого же игрока. Здесь я столкнулся со следующей проблемой. Когда создаётся копия префаба, то он становится самостоятельным объектом, т.е. когда звездолёт улетает от места выброса щита, то щит остаётся на месте, вместо того, чтобы защищать корабль. Тогда я определил щит как **дочерний объект** корабля, - щит стал повторять движения игрока. Но когда в него попадал снаряд, то HP игрока отнималось (ведь щит теперь часть корабля игрока). Чтобы решить эту проблему, я воспользовался **тегами**. Определил **тег Protect**, прописал этот тег в свойствах объекта-щита (пуля не будет отнимать HP у игрока, если попадёт в любой объект с тегом Protect).

Рис.6. Появление щита

При достижении **life** (HP) от 60 до 30 появляется дым, ниже 30 – огонь. Если life = 0, то звездолёт красиво разрушается с использование спецэффектов, и выходит сообщение с именем победителя. На рис.7. представлено сообщение,

которое выйдет, если победит игрок по имени Player2.

Рис.7. Сообщение о победе для Player2.

Движок Unity 3D позволил мне быстро и легко создать данную игру. Я лично убедился, что этот движок позволяет создавать любые компьютерные игры, какие только можно себе представить. Единственное ограничение, которое может быть – это фантазия разработчика.