Contents

[Добавление карты 1](#_Toc160214370)

[Создание карты 1](#_Toc160214371)

[Генерация элементов карты 1](#_Toc160214372)

[Элемент карты 1](#_Toc160214373)

[Сущность. 2](#_Toc160214374)

[Интеграция функции создания карты 2](#_Toc160214375)

[Создания «События». 2](#_Toc160214376)

[Глобальные переменные 3](#_Toc160214377)

[Интеграция события 6](#_Toc160214378)

# Добавление карты

В данном документе будет описан процесс создания и добавления новой карты. А также написания функции «события» для этой карты.

# Создание карты

Карта создаётся посредством выполнения соответствующей функции в файле *GameMap.cpp*. Функция должна иметь вид:   
void Game::CreateMapN(Vec2F\* ships\_positions, float\* ships\_angles);

N в названии заменяется на номер кары.

ships\_positions -- массив позиций игроков, его длинна всегда равна 4. Используется для задания позиций игроков

ships\_angles -- массив направлений игроков, его длинна всегда равна 4. Используется для задания направлений игроков

Прототип функции надо указать в файле Game.h, ниже уже существующих прототипов. В комментарии к прототипу надо указать название карты.

Если для создания карты потребуется дополнительная функция, то такая функция должна иметь название void Game::CreateMapN\_<имя функции>. Её реализация должна быть помещена под функцией CreateMap, а прототип над ней в том же файле.

Создание карты можно разделить на 2 части:

1. Генерация элементов карты
2. Генерация сущностей на карте

# Генерация элементов карты

Объект «карта» включает в себя три массива:

1. Массив прямоугольников
2. Массив окружностей
3. Массив ломаных линий

Каждый из массивов требуется инициализировать, заполнить и только после этого передать в функцию.  
После переменной массива должно следовать количество элементов в массиве.

Пример функции создания карты.  
map.Set(recs, recs\_count, circs, circs\_count, polys, polys\_count);

# Элемент карты

Элемент карты «MapElement» основной класс от которого наследуются описанная выше троица. Класс описан в файле Map.h. Хотя его конструктор и не используется при создании карты, следует обратить внимание на поле EngineTypes::Map::property\_t properties; Данное поле задаёт свойства элемента по отношению к сущностям, в том числе и игрокам. Элемент имеет четыре параметра: 1) UNBREACABLE 2) KILLER 3) AGRESSIVE 4) CLOSED. Каждый из параметров может быть включён или выключен изменением соответствующего бита в поле «properties». Маски полей и их описания указаны в файле GameEngineConstants.h. Конструкторы объектов прямоугольника, круга и ломаной линии с описаниями находятся в файле Map.h.

# Сущность.

Сущности описаны в файле Entity.h.

Для добавления сущности во время создания карты надо вызвать функцию AddEntity(Объект сущности).

За один вызов можно добавлять только одну сущность. Прямое обращение к памяти через массивы ЗАПРЕЩЕНО! Функция AddEntity, вместе с другими функциями доступа к памяти находится в файле Game.h.

# Интеграция функции создания карты

После написания функции CreateMap, её надо интегрировать в игровой движок. Для этого необходимо выполнить следующие действия:

1. В файле GameRealisationConstants.h в области «map pull» надо добавить строку #define MAP\_<имя карты> Nu, где N – номер карты, указанный в названии функции CreateMapN.
2. В том же файле в области match rules надо увеличить значение GAME\_MAPS\_COUNT на 1.
3. В файле Game.cpp, в реализации функции void Game::InitLevel функции надо добавить новый case по аналогии с уже добавленными.
4. В том же файле в реализации функции void Game::InitMenus() в области «maps select menu» надо добавить «buttons[MAP\_<имя карты>].SetText("<имя карты>");». Возможно потребуется изменить размер шрифта, для этого надо написать строку «buttons[MAP\_<имя карты>].text\_size = <размер текста>;»

# Создания «События».

Событие – это функция, которая будет вызываться каждый раз при пересчёте всех объектов, то есть примерно 100 раз в секунду. Внутри события может происходить что угодно, кроме того, что может вызвать критическую ошибку в программе. Таким образом в функции события нельзя:

1. Делать прямую запись в память. То есть создавать или удалять объекты без помощи функций CreateEntity, RemoveEntity, DestroyEntity.
2. Удалять пилотов и корабли.
3. Изменять значение счётчика. Но можно создавать запросы на изменение, которые будут учтены в конце матча.
4. Обращаться к памяти напрямую, предварительно не заблокировав соответствующий mutex.

Событие описывается в функции вида void Game::EventN(). Реализация и прототип размещаются по аналогии с CreateMap.

# Глобальные переменные

Событие имеет доступ на чтение к глобальным переменным. Защита от записи отсутствует, однако изменение некоторых глобальных переменных может поломать программу. Ниже представлен список глобальных переменных доступных для функции события ТОЛЬКО НА ЧТЕНИЕ.

//Global tic value. Not use by functions.

GameTypes::tic\_t global\_timer;

//Global tic value when inverse flag was change value.

GameTypes::tic\_t rotate\_inverse\_chage\_tic;

//Global tic value for use in functions.

GameTypes::tic\_t end\_match\_tic;

GameTypes::tic\_t stuning\_timer;

//Game status true = The game is on. false = The game stops.

bool play\_round;

//Game status true = The game start next raund.

bool play\_match;

//Game status true = The game is paused. false = The game is on.

bool pause\_round;

//Index of array is ship's player number. Values of array are numbers of teams by every ship.

//Array changed in Game::Init::Menus.

GameTypes::entities\_count\_t\* teams;

//Index of array is ship's player number. Values of array are numbers of teams by every ship.

//If player not play in current level, then alue is zero.

GameTypes::entities\_count\_t\* playing\_teams;

//Index is team number

//Value is count of exist players in team

GameTypes::players\_count\_t\* players\_in\_team;

GameTypes::maps\_count\_t selected\_maps\_id\_array\_length;

//Controle is inversing

bool rotation\_inverse;

//Buttons positions in inveres.

bool buttons\_inverse;

//This flag will be active after the completion of the MatchInit function and inactive after the end of the match.

bool flag\_all\_entities\_initialisate;

//Function CheckEndMatch is active.

bool flag\_update\_end\_match;

//If true match will finish.

bool flag\_end\_match;

//If true results of round are drawind.

bool flag\_round\_results;

GameTypes::maps\_count\_t current\_event;

//Current scores of players.

GameTypes::score\_t\* scores;

//Last scores of plauers.

GameTypes::score\_t\* last\_match\_scores;

//Id of the current map of the level.

GameTypes::maps\_count\_t current\_map\_id;

//If player will get end\_match\_score or high scor then this people will become potencial winner.

//This variable is setted by InitMatch function.

//This variable chaned in NextLevel function.

GameTypes::score\_t end\_match\_score;

//Count of annihilation area generators on the map.

GameTypes::entities\_count\_t annih\_area\_gens\_count;

//Count of deceler areas on the map.

GameTypes::map\_elements\_count\_t deceler\_areas\_count;

//Count of gravity generators on the map.

GameTypes::map\_elements\_count\_t grav\_gens\_count;

//Count of players on the map.

GameTypes::players\_count\_t players\_count;

//Count of ships on the map.

GameTypes::players\_count\_t ships\_count;

//Count of pilots on the map.

GameTypes::players\_count\_t pilots\_count;

//Count of knifes on the map.

GameTypes::entities\_count\_t knifes\_count;

//Count of lazers on the map.

GameTypes::entities\_count\_t lasers\_count;

//Count of bombs on the map.

GameTypes::entities\_count\_t bombs\_count;

//Count of turrets on the map.

GameTypes::map\_elements\_count\_t turrets\_count;

//Count of map's lazers on the map.

GameTypes::map\_elements\_count\_t mega\_lasers\_count;

//Count of asteroids on the map.

GameTypes::entities\_count\_t asteroids\_count;

//Count of bonuses on the map.

GameTypes::entities\_count\_t bonuses\_count;

//Count of bullets on the map.

GameTypes::entities\_count\_t bullets\_count;

//Count of particles on the map.

GameTypes::entities\_count\_t dynamic\_particles\_count;

//Count of particles on the map.

GameTypes::entities\_count\_t particles\_count;

//Count of portals on the map.

GameTypes::map\_elements\_count\_t portals\_count;

//Bonus in all ships at the start of the game.

EngineTypes::Bonus::inventory\_t start\_bonus;

//This variable stores all the values of rules of ghe game. To change the values of rules of ghe game use |= and &=. To get the values of rules of the game use &.

GameTypes::game\_rules\_t game\_rules;

//game variables

//game lists

//Array of flags store data about shoots of ships in current tic.

bool\* ships\_shooting\_flags;

//Controled entities refer to this array.

bool\* shoot\_flags;

//Controled entities refer to this array.

bool\* rotate\_flags;

//Tumer for check double clk for ships' burnout.

GameTypes::tic\_t\* double\_clk\_timers;

//Controled entities refer to this array.

bool\* burnout\_flags;

//If value in the array then ship can shoot.

GameTypes::tic\_t\* ships\_can\_shoot\_flags;

//This array contains flags for maps that will be used in the current match.

bool\* map\_pull\_array;

//This array contains maps' ids that will be used in the current match.

GameTypes::maps\_count\_t\* selected\_maps\_id\_array;

//This array contains objects that will be spawned in the current match.

bool\* object\_pull\_array;

//This array contains bonuses, buffs and rules that will be spawned in the current match.

bool\* bonus\_pull\_array;

EngineTypes::Bonus::inventory\_t inventory\_template\_mask;

//game lists

//game objects

Logs logs;

//Array of shiips.

Ship\* ships;

//Array of pilots.

Pilot\* pilots;

AnnihAreaGen\* annih\_area\_gens;

//Array of asteroids.

Asteroid\* asteroids;

//Array of bonuses.

Bonus\* bonuses;

//Array of bombs.

Bomb\* bombs;

//Array of bullets.

Bullet\* bullets;

//Array of dynamic particles.

DynamicParticle\* dynamic\_particles;

//Array of knifes.

Knife\* knifes;

//Array of lazers.

Laser\* lasers;

//Array of particles.

Particle\* particles;

//Object stores data about map on current level.

Map::MapData map;

//Array of deceleration areas.

DecelerationArea\* deceler\_areas;

//Array of gravity generators.

GravGen\* grav\_gens;

//Array of turrets.

Turret\* turrets;

//Array of lasers of the map.

MegaLaser\* mega\_lasers;

//Array of porttals

Portal\* portals;

Camera camera;

Для сохранения значений между вызовами событий надо использовать *статические поля*.

Также для для более быстрых расчётов рекомендуется использовать функции из Mat.h и Entity.h

# Интеграция события

После реализации события его надо интегрировать. Для этого необходимо в файле Game.cpp, в функции void Game::PollEvents() добавить строку «case MAP\_<имя карты>: EventN(); return;»

Событие можно добавлять ТОЛЬКО ПОСЛЕ добавления самой карты.

# Дебаг

Так как для внесения изменений требуется полный перезапуск программы, то для упрощения проверки карты рекомендуется в файле Game.cpp в функции void Game::InitLevel() добавить под строкой current\_map\_id = GenerateRandomMapId(); добавить строку std::cout << "void Game::InitLevel()::current\_map\_id=" << (current\_map\_id = <map number>) << std::endl; файле за место «map number» указать номер карты.