

uRTS toolkit

v3.3

Contact: Chanfort

Email: chanfort48@gmail.com

20/09/2015

System Requirements

Unity 4.6, 5.0, 5.1 or 5.2.

Introduction

uRTS Toolkit is highly advanced system to ease RTS genre game creation and development. It creates massive battles (thousands of units) with typical elements of strategy games (eg. resources, buildings, diplomacy). The kit is designed to have amazing performance, good frame rate and speed in our games, with the 3DSprites system (converts 3D models to textures, which then will be our soldiers) and optimized battles system script (see technical details below).

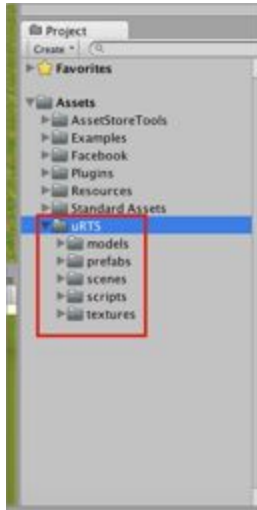
Features:

- Easy navigation (Navmesh)
- Units grouping and formation system
- Battle management AI
- Town growth AI
- Wonderer AI
- Diplomacy system and decision AI
- RTS Camera Movement
- Units selection and control
- Realistic archery
- 3DSprites creator and controller
- Battle statistics
- Basic Animated models
- Selection marks and health bars
- UI 4.6 menus
- Basic Economy
- Procedural terrain and forest generation toolkit
- Resource spawning and collection system
- 11 building and 4 animated unit models, made in Blender
- Pure C# source code
- Unity 5 support

Start in uRTS

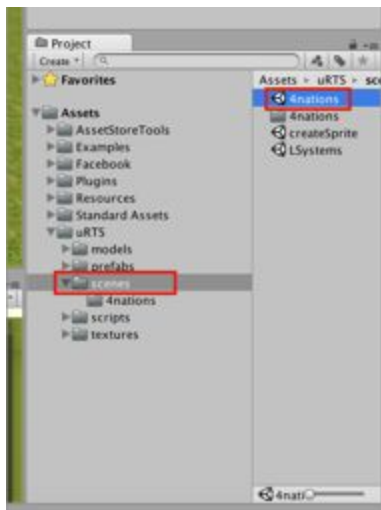
This section will explain how to use pre-build game example of uRTS toolkit. To run it just follow these steps.

1. Create new project in Unity and import uRTS toolkit package. Before moving to next step make sure that you see the same folder structure as shown in the screenshot below:

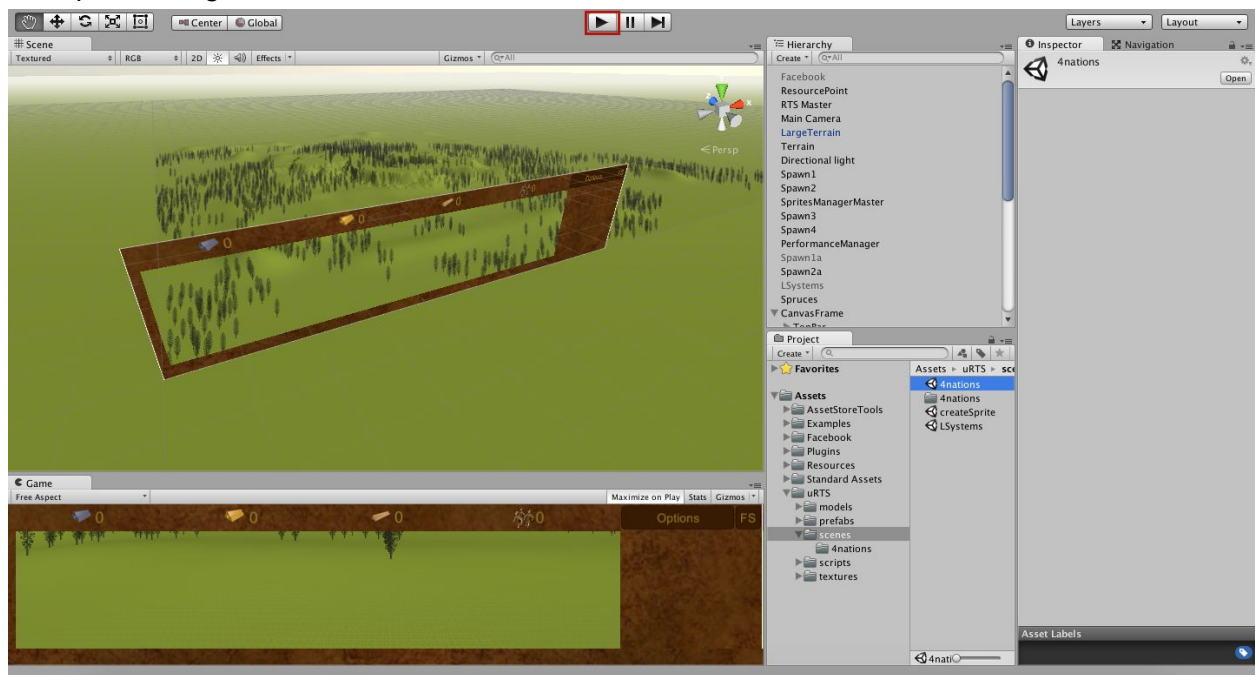


2. If you see all folders, marked with red rectangle, you can continue, if not, check if you downloaded this asset and if it is placed in your project assets folder.

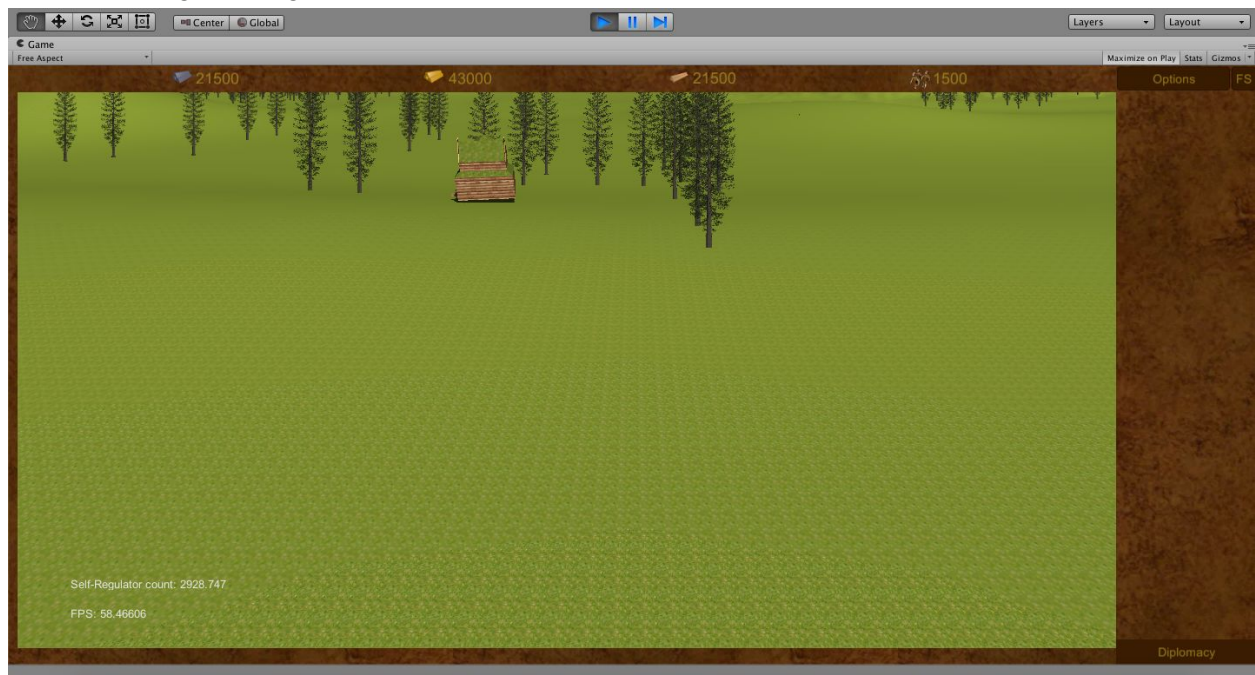
3. Open pre-build scene by entering to “scenes” subfolder and double clicking “4nations” scene file



4. You should see in Editor now something similar as shown below. Click “run” to see pre-build example running



5. Successfully running build should show units and buildings placed on terrain, like in this screenshot. You can start now playing (e.g. select your castle, click eye icon in right panel to start spawning building, etc.).



Set-up on empty scenes

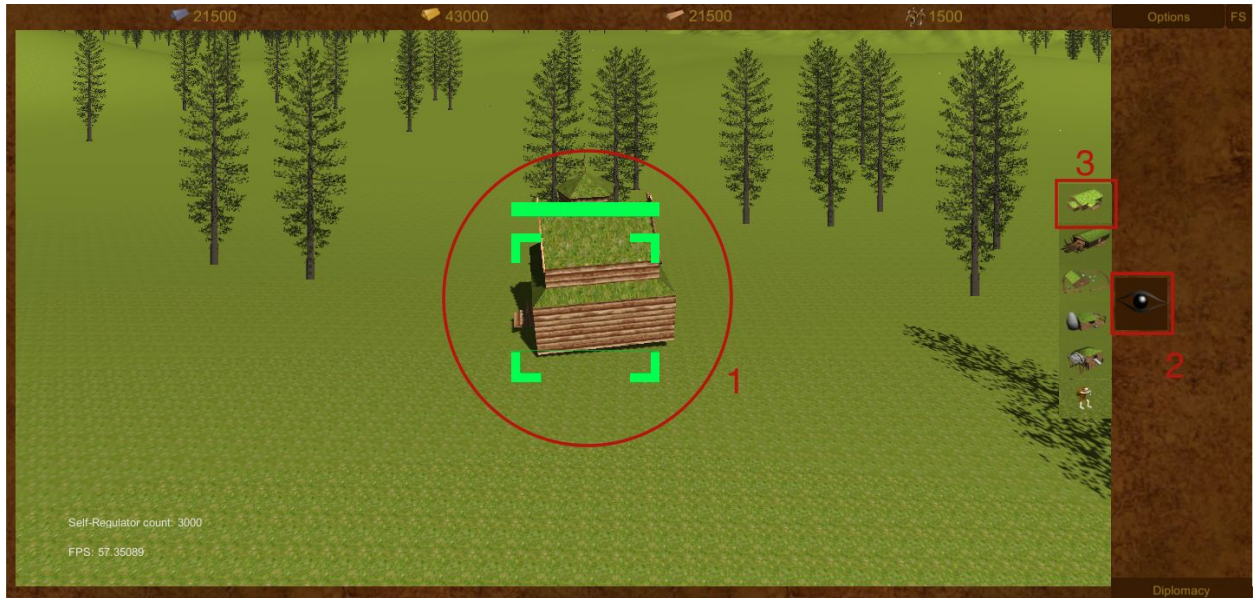
uRTS uses easy set-up on empty scenes system. It can be done through editor script “RTSCompiler.cs”, show below:



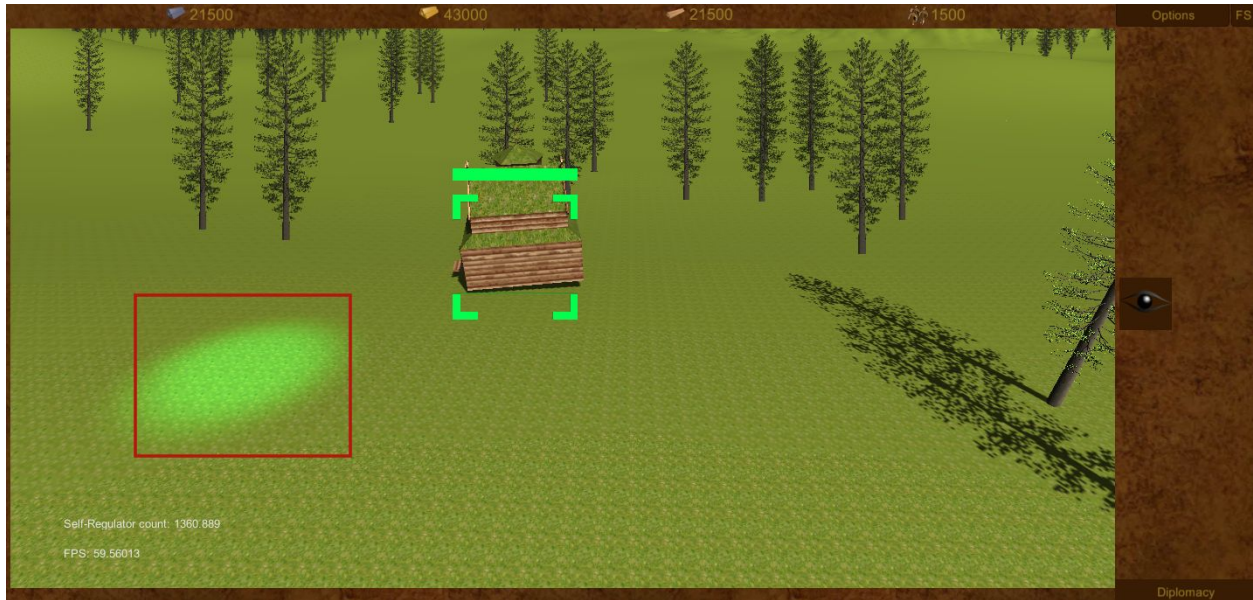
Once new scene is open, before continue it is needed to add terrain and bake NavMesh on the terrain manually. To use RTSCompiler, create new gameObject and attach RTSCompiler script. The screenshot above shows how parameters in RTSCompiler are set for default scene. Currently the order of visible parameters is not very flexible and only more experienced users should experiment with them in order to avoid unexpected behaviour, errors and crashes in uRTS. RTSCompiler is created to ease setting up uRTS toolkit on empty scenes. If behaviour becomes different from original set-up the first thing to do is to check manually how scripts and gameObject are set in original scene. To compile uRTS on empty scene, click “Generate” button, which will create all needed gameObjects and sets script linking correctly. You can also click “Clean up” in order to uncompile uRTS from the scene.

Simple gameplay

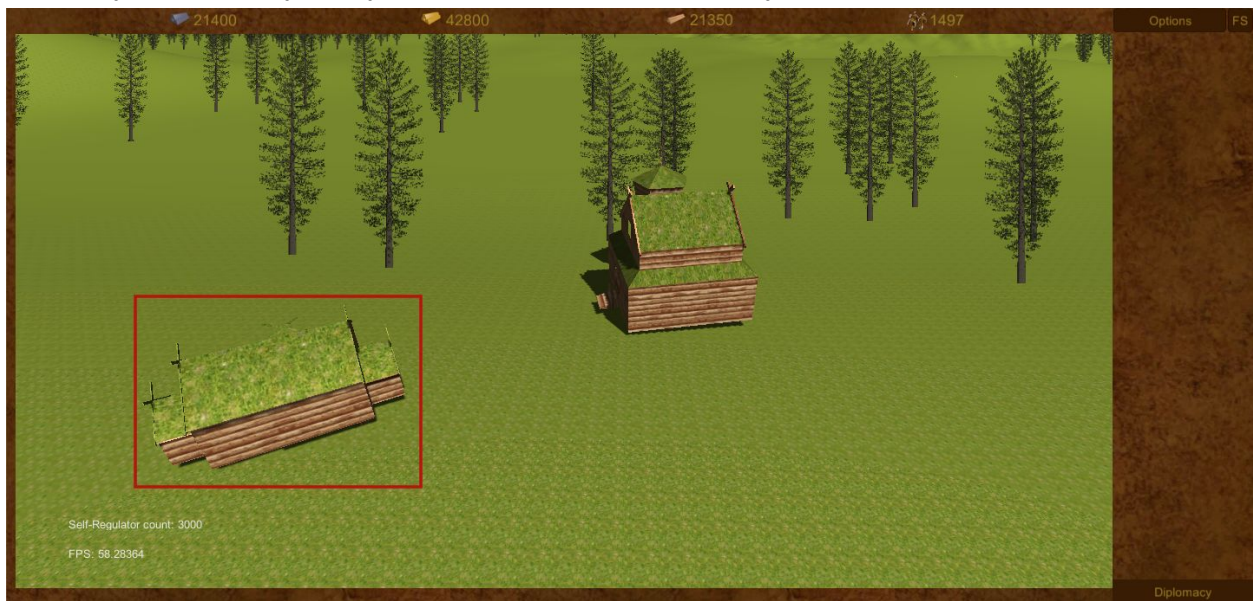
1. Player starts the game with single building - his castle. Left click on your castle (1) and “eye” button in right hand side will appear (2). Left click on “eye” button to see what else you can build. Start building barracks first, where you can spawn military units. Left-click on barracks icon to start building them (3).



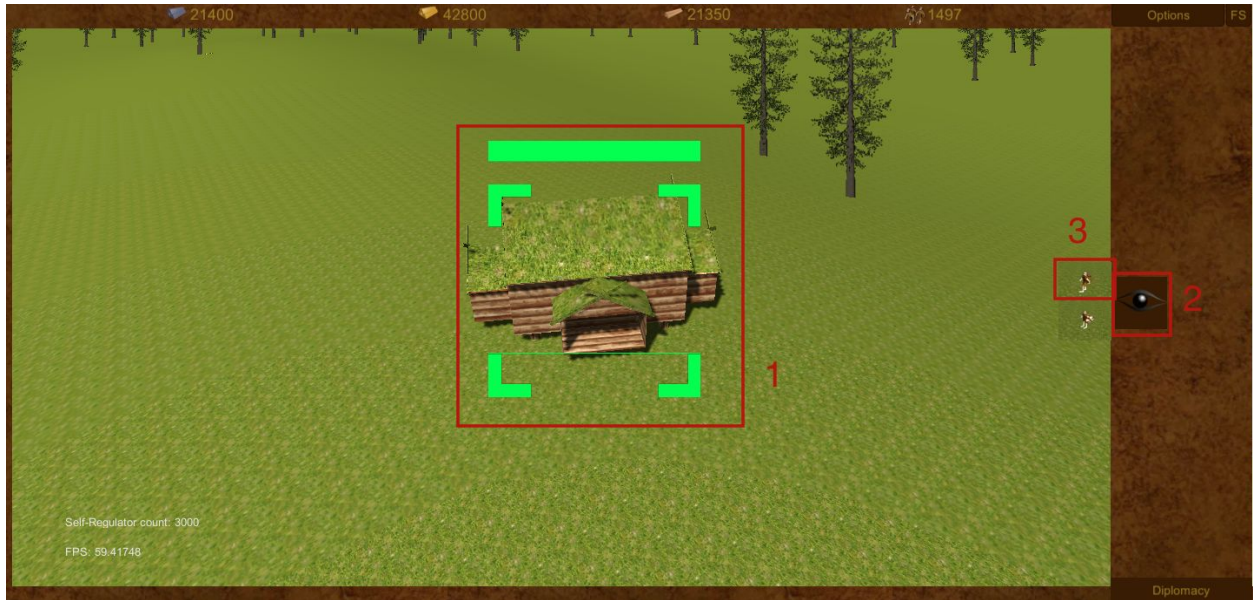
2. Move your mouse around to see glowing circle on the ground - this is the phase when you can chose where to build your building. The circle will be green if building is allowed or red if not in the location, where mouse is pointing. You can't build buttons in several locations:
 - a. too far away from your base
 - b. steep slopes in mountains
 - c. on places where other buildings stay
 - d. locations of trees



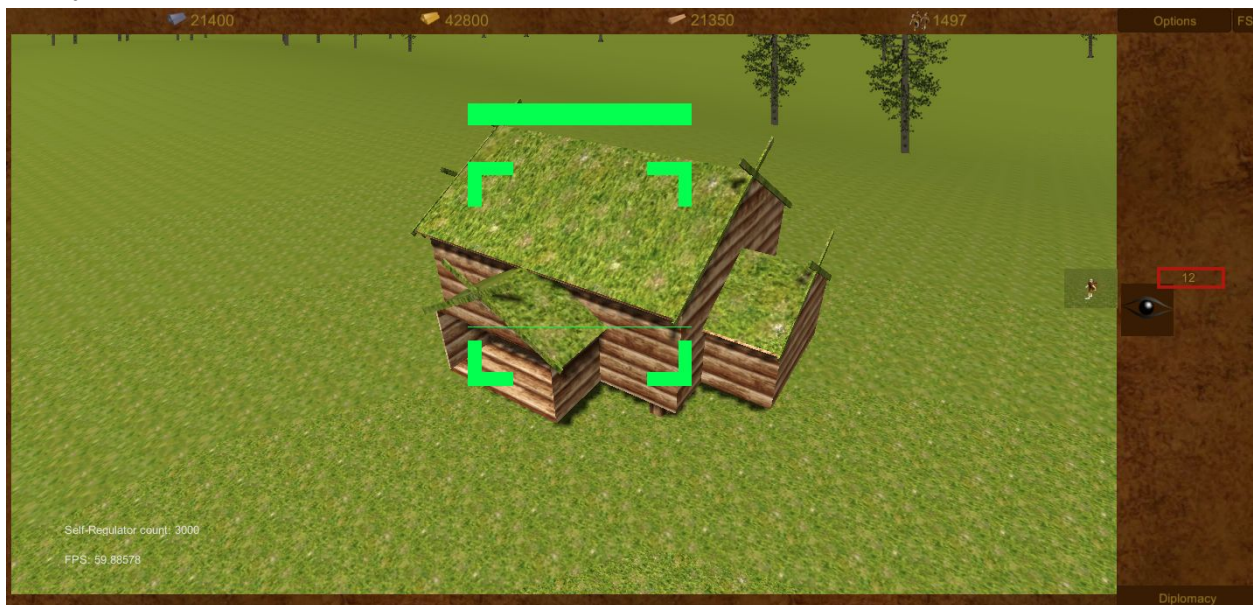
3. Once you are happy with your location, left click to build your barrack.



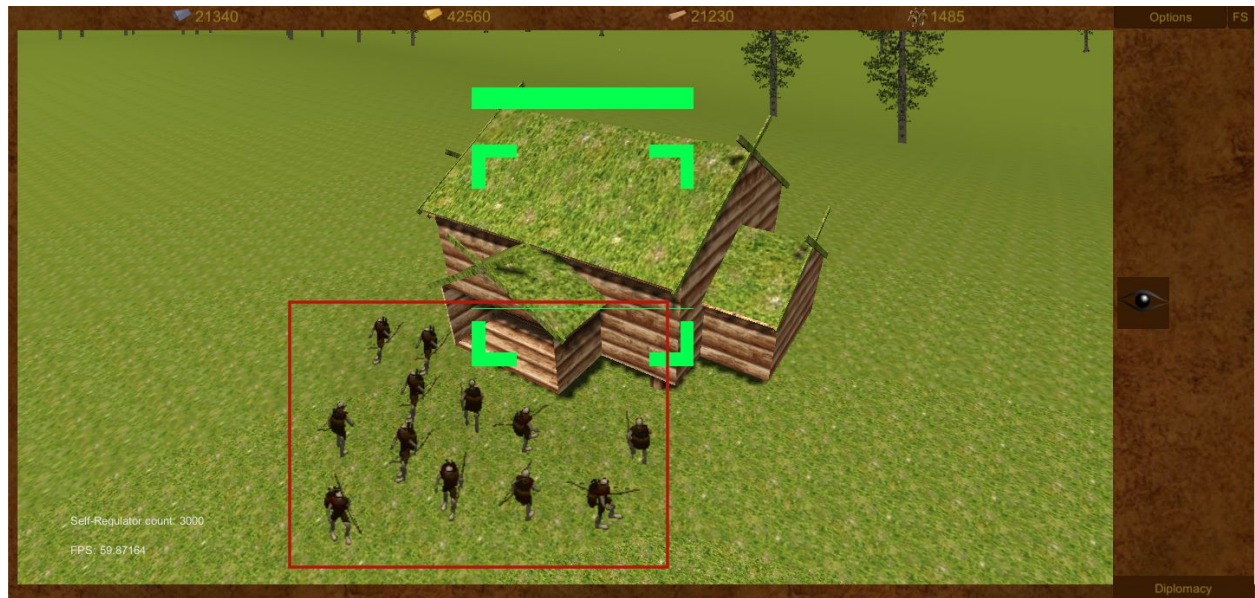
4. Left click on barracks to see what units you can spawn at the moment.



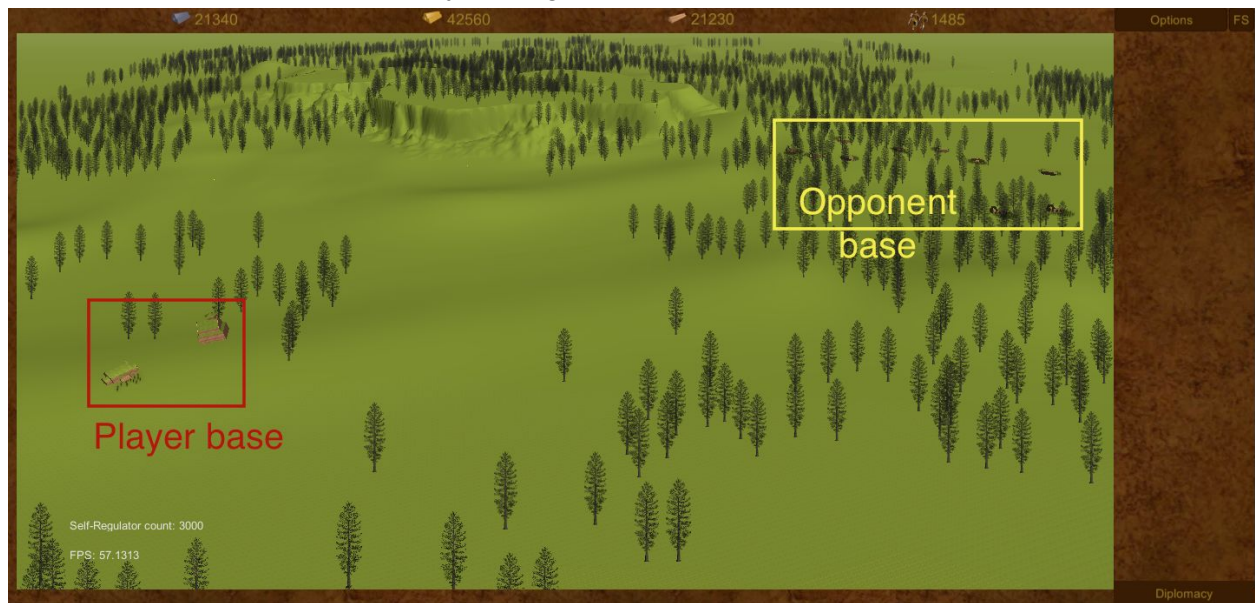
5. When you chose which type of units you want to spawn, left click on icon. The counter in the right hand side will appear. Now you can scroll with mouse to increase/decrease the number of units you want to spawn.



6. Finally, left click again on unit icon to start spawning. Units will start to appear near the entrance of your barracks.



7. You can now look around for other nations on the map, which you can challenge. Zoom out to see whole map and look where is your target base.



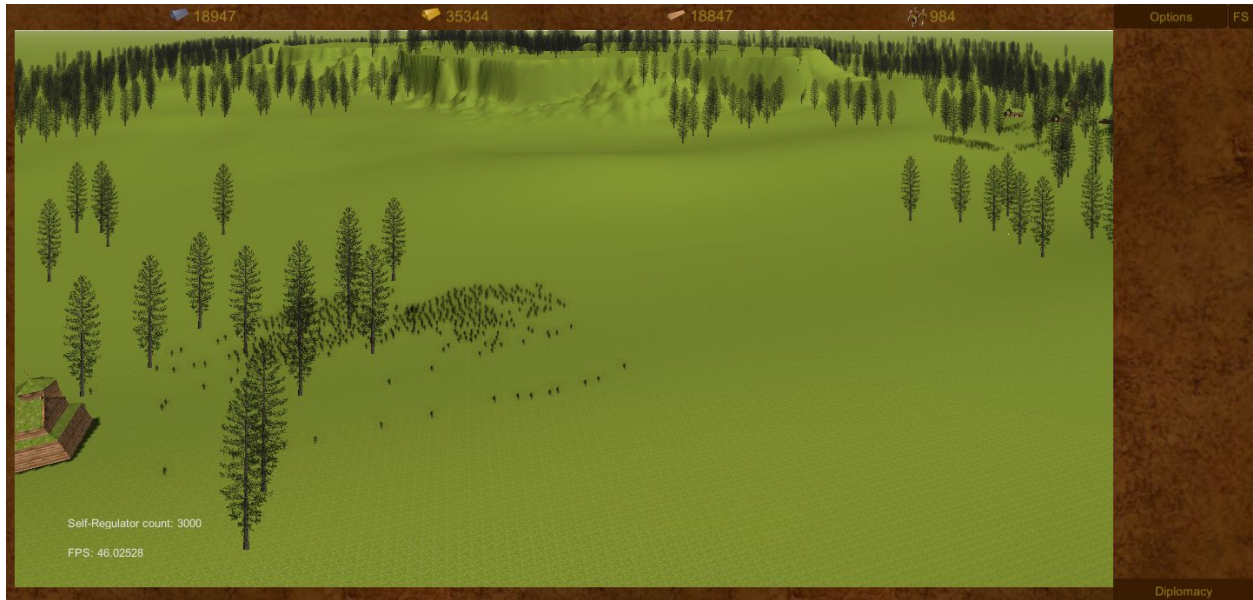
8. Once you build army large enough to challenge your opponent, left click diplomacy to see nations menu. Chose the nation which you want to make a dialog



9. In the dialog click “.. prepare to die” option to declare a war.



10. Finally your troops will start swarming towards enemy base and enemy will send their troops towards you.



11. When both armies will clash, the fight begins.



By default fights can hold up to several thousands units. Larger fights usually leads to low FPS rates. There are no other limits than FPS rates for sizes of troops in uRTS.

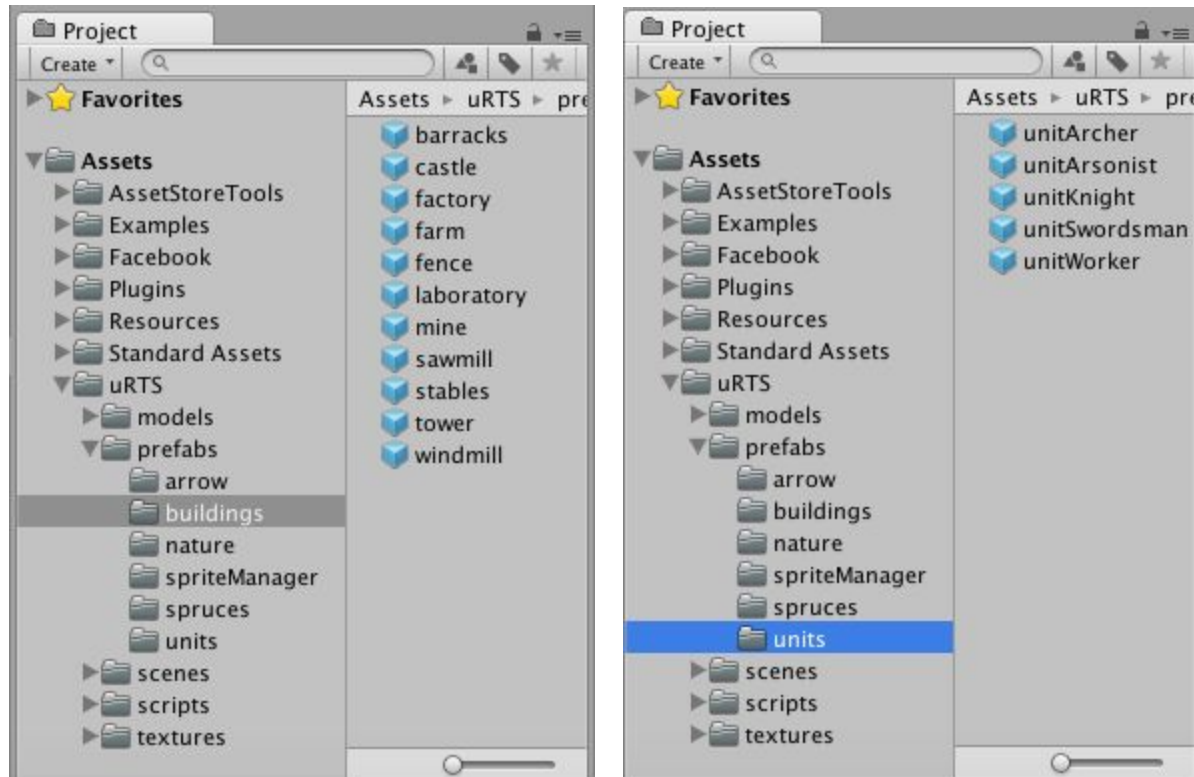
Parameters

We looked at the details how to play and run with default content in uRTS. Now we will take a look what we can change in order to better adapt uRTS to your own project. Inspector parameters will allow you to customize uRTS system for your needs. However, more specific

applications may involve modifying C# scripts, which will not be discussed in this documentation.

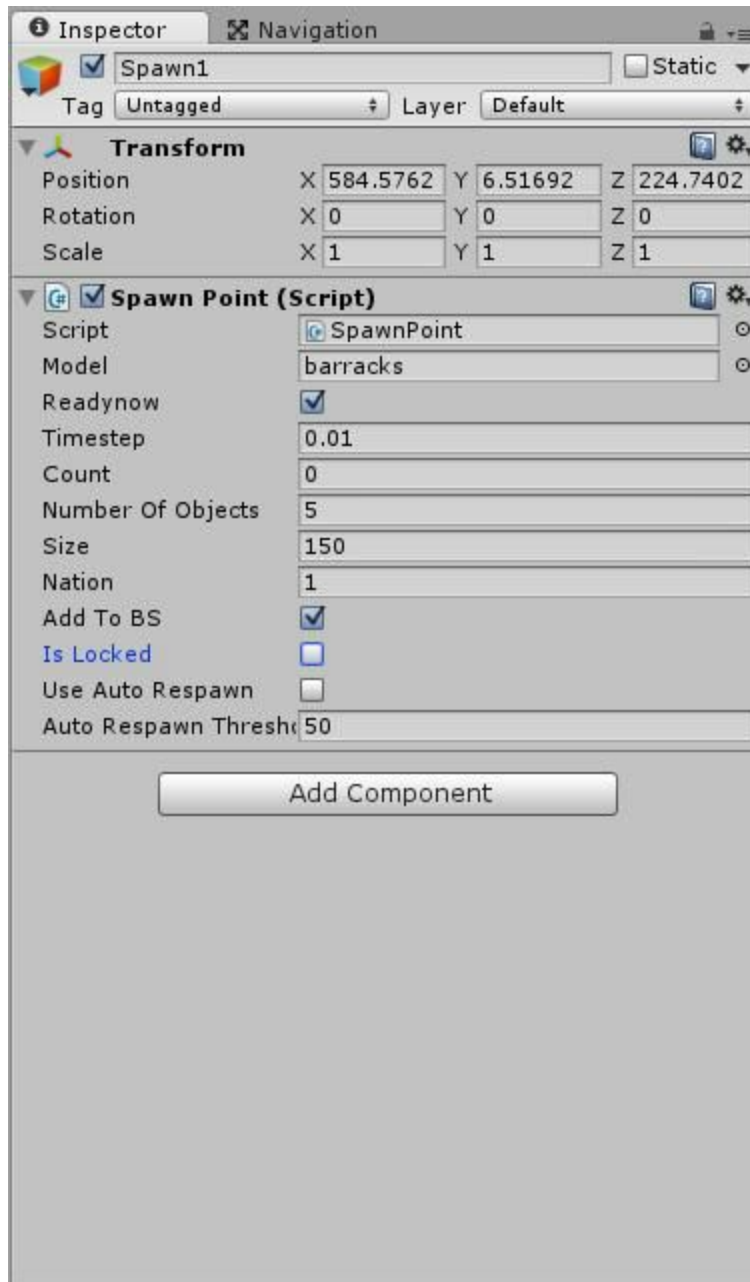
Building and unit prefabs

Default buildings and units in uRTS are saved into form of GameObject prefabs. You can find them in “Assets/uRTS/buildings” and “Assets/uRTS/units” directories respectively. There are 11 building and 5 unit prefabs in uRTS.



Each prefab usually has NavMeshAgent and UnitPars.cs script attached. Some buildings (e.g. Castle, barracks) has SpawnPoint.cs scripts attached, which are responsible for spawning corresponding units. Units usually has SpriteLoader.cs script attached, which is responsible for showing these units as 3dSprites.

SpawnPoint



1. **"Model"**: Game object to be spawned.
2. **"ReadyNow"**:
3. **"Timestep"**: spawns object every x seconds, where x is number, assigned for this parameter.
4. **"Count"**
5. **"Number of Objects"**: the number of objects which "Spawn Point" will create.
6. **"Formation Size"**: the size of Formation for units to spawn.
7. **"Size"**: the size of the box in space, where objects will be spawned
8. **"Nation"**: the nation of unit.
9. **"Add to BS"**:

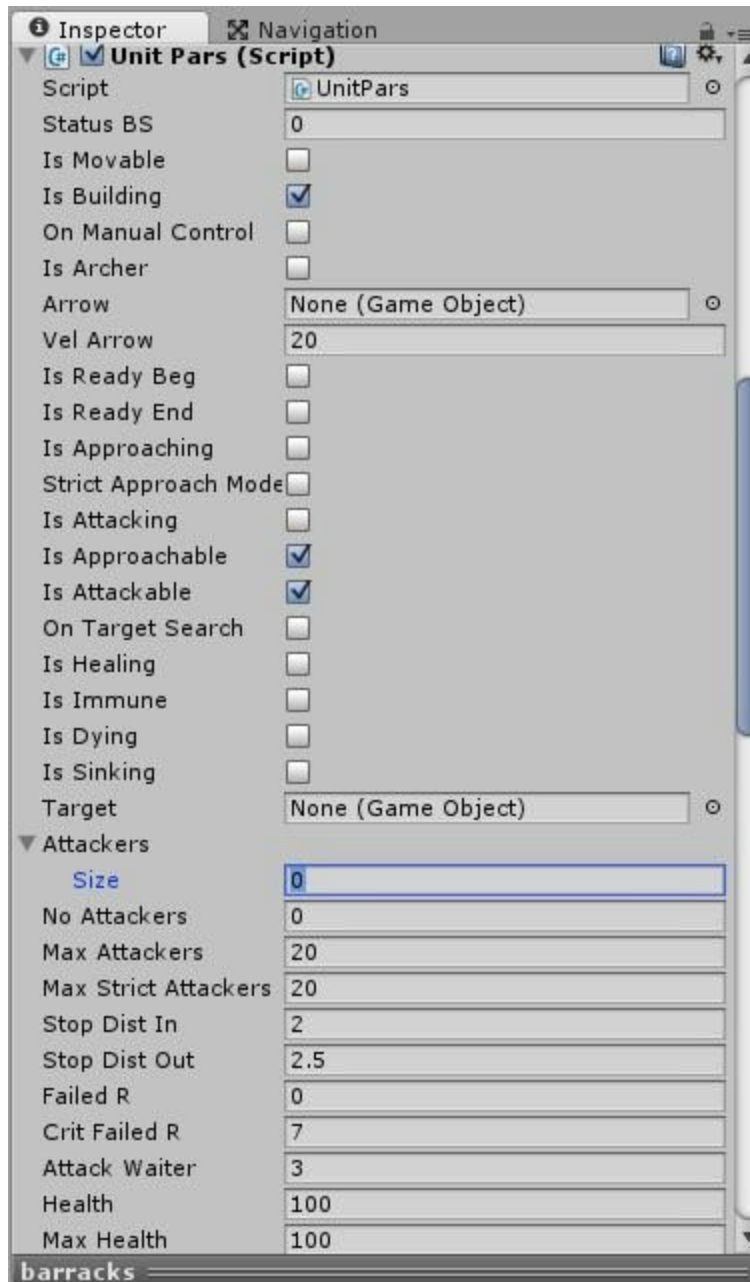
10. **"Is Locked"**

11. **"Use auto respawn"**: Used to automatically respawn other nation units if nation goes low in number

12. **"Auto Respawn Threshold"**: number of units to respawn.

UnitPars

UnitPars is the main script containing most of unit parameters. Most of parameters here changing very rapidly from version to version, as uRTS progresses, however the idea of UnitPars is to hold RTS parameters of specific units in the unified way. UnitPars are closely linked to RTSMaster.cs script, which holds all unit types in the asset, while UnitPars characterises properties of individual units.



1. **"Is Moveable"**: boolean value, which shows that unit is moveable
2. **"Is Building"**: boolean value, which shows that unit is building
3. **"OnManualControl"**:
4. **"Is Archer"**: boolean value, which shows that unit is archer.
5. **"Arrow"**: Model of the arrow
6. **"VelArrow"**: Velocity of arrow
7. **"Is ready Beg"**:
8. **"is ready End"**:

9. **"Is Approaching"**:boolean value, which shows that unit is approaching to their target
10. **"Is approachable"**:boolean value, which shows that unit is approachable
11. **"is Attackable"**:boolean value, which shows that unit is attackable
12. **"On target search"**
13. **"Is Healing"**:boolean value, which shows that unit is healing
14. **"Is Immune"**:boolean value, which shows that unit is immune (never die)
15. **"Is Dying"**:boolean value, which shows that unit is dying
16. **"Is Sinking"**:boolean value, which shows that unit is sinking
17. **"Target"**: shows the target of the unit
18. **"Attackers"**
19. **"No Attackers"**
20. **"Max Attackers"**:number of attackers, allowed to attack this unit
21. **"Max strict attackers"**:
22. **"Stop Dist In"**
23. **"Stop Dist Out"**
24. **"failedR"**
25. **"Crit Failed R"**:when approachers are approaching their targets, distance between them must decrease. If not – it is counted as failure to approach. This parameter set maximum allowed failures before resetting new target.
26. **"Attack Waiter"**
27. **"Health"**:unit current health – if it drops below 0, unit dies.
28. **"Max Health"**:maximum health, which unit can hold.
29. **"Self Heal Factor"**:amount of health, which unit automatically receives per time unit
30. **"Strength"**:strength of attacker – factor at which attacker removes opponent health.
31. **"Defence"**:possibility to avoid attacker damage.
32. **"Max Death Calls"**
33. **"Max Sink Calls"**
34. **"Nation"**:the nation of unit.
35. **"Is Selected"**:boolean value, which shows that unit is selected with LMB
36. **"Preparing Moving MC"**
37. **"Is Moving MC"**
38. **"Is On BS"**:boolean value, which shows that unit is added on BattleSystem
39. **"crit Failed Dist"**
40. **"Manual Destination"**
41. **"ReEnclosed"**
42. **"AnimationToUse"**
43. **"Cost Iron"**
44. **"Cost Gold"**
45. **"Cost Lumber"**
46. **"Cost Population"**

Terrain

The terrain is where we added the most important script in uRTS toolkit: BattleSystem.cs
There are also added other scripts here:

- Diplomacy
- Scores
- Selection Mark
- Economy



Since version 3.3, uRTS supports procedural terrain generation and usage. The usage is currently limited that only on the main terrain units can be built, while other terrains are just cosmetic.



Battle System

“**BattleSystem.cs**” is responsible for simulating battles. It moves attackers towards their targets, fights targets, deals with unit health and controls dying units until they are safely unset before finally being destroyed. The script has only one supported parameter at the moment: “Attack Distance”, which sets the maximum allowed distance between units to be used.

Diplomacy

“Diplomacy.cs” is an additional element in uRTS, which allows to set up peace or war between different nations in the scene.

The script itself has only three parameters:

1. “**Number Nations**”: the number of nations, available in scene:
2. “**Player Nation**”: nation ID of player:
3. “**War notice warning**”:

Higher level AI system

4 main AIs in uRTS are responsible for controlling non-player based nations. These AIs take economical, military and diplomatic decisions and are also responsible for balance between these decisions.

Battle AI is responsible for searching and dynamically assigning targets to military units. AI searches for neighbour targets and assigns them to attackers. Search is based on KDTree algorithm, so is fast enough even to reassign targets dynamically when units are moving on the map and new targets becomes closer than previously assigned ones.

WondererAI sets units to visit neighbour nations. Each nation is sending small amount of it's military units in order to see what neighbours are doing and better prepare for war if threat is upon the sending nation.

Town development AI is responsible for what buildings and how many of them are built in the town. It also controls what units and how much of them are being spawned from barracks. This AI also controls workers in order to collect resources needed for development.

Diplomacy AI decides when to declare a war, to beg for mercy or to enslave neighbour nations.

Scores

Scores are used to count number of buildings/units, damages and some other scores for individual nations. To access some statistics press “t” two times in the keyboard while in game.

Economy

Economy controls player and other nations economies. Once player or other nations run low in resources, they can't make new buildings/units.

Archery

uRTS supports archery units. There are made prefab in Units folder. Archery parameters in UnitPars.cs are:

1. “**Is Archer**”: boolean value, which shows that unit is archer.
2. “**Arrow**”: arrow model used for arrows.
- 3.. “**Vel Arrow**”: arrow shooting velocity.

Arrows are shot with initial conditions and their trajectory is not modified anymore - they are only flying in gravitational field. Once arrow moves bellow the terrain, it is destroyed.

Calculations of initial shooting angle (with fixed velocity "Vel Arrow") are calculated in 3 approximations:

- a. shot on flat terrain with static target;
- b. shot uphill/downhill with static target;
- c. shot uphill/downhill with moving target.

a. and b. approximations are working very well. However c. approximation only sometimes seems to be working correctly, as it is not completely independent from a. and b. approximations. This requires further testing.

Script currently runs with all three approximations used.

There is always calculated distance between arrow and its target. If distance becomes smaller than target enclosed sphere, target obtains damage, which is carried with arrow.

Basic models

Basic animated models are included in this asset for testing purposes at the moment. Improvements, based on these tests should bring animated characters for uRTS in future releases.

Archer model



Knight model



3DSprites

3dSprites is a system, allowing to create and manipulate 2d sprites from any kind of 3d animated objects in such way, that 2d sprites looks like 3d objects. 3dSprites creation and usage in the asset are the main two steps, which should be taken to convert 3d objects into animated 3dSprites. Both processes use high-end user friendly parameters. Creation class is named as SpritesCreator.cs and usage class - SpriteLoader.cs.

SpritesCreator.cs input parameters are:

- “**model**”: animated object to be animated;
- “**modelName**”: the name of model, which will be animated;
- “**animationName**”: the name of animation; note, that this should match with animation name, which actual model has.
- “**objectSize**”: size of object.
- “**spriteResolution**”: final resolution of sprite tile (in pixels).
- “**numberOfColumns**”: the number of columns in sprites sheet texture.

“numberOfFrames”: resolution of frames per second for sprite animation.

“horRotLevels”: number of horizontal rotation levels to be modelled.

“verRotLevels”: number of vertical rotation levels to be modelled.

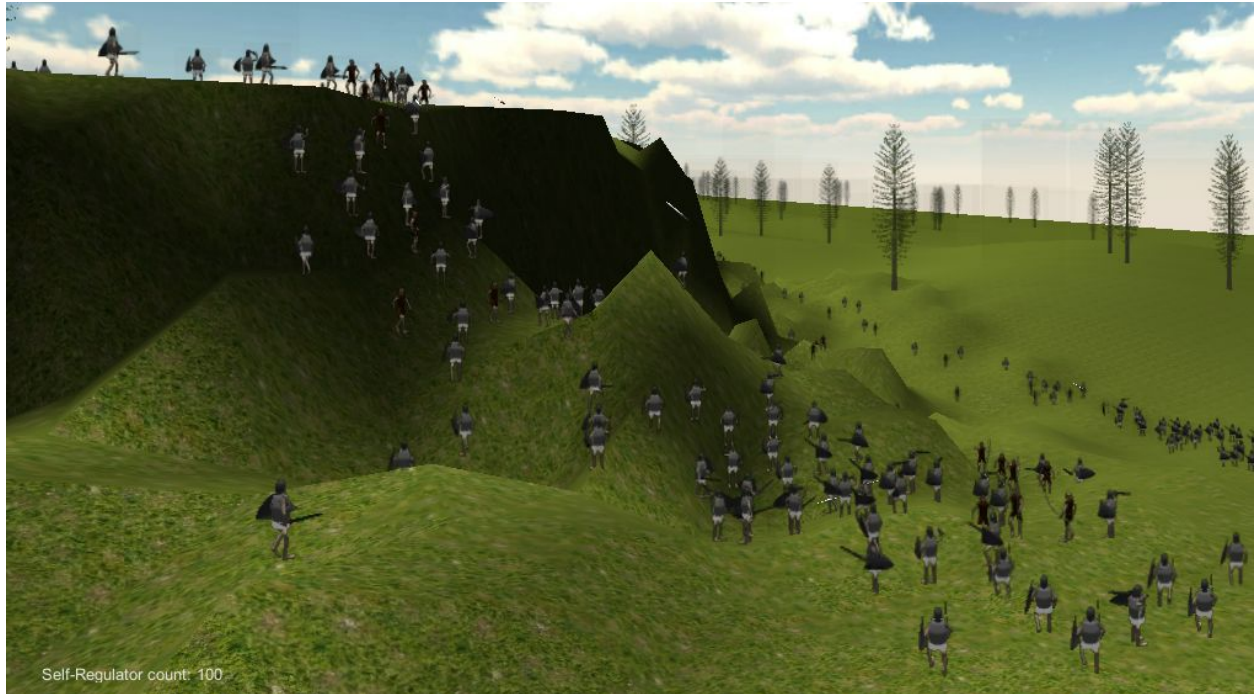
Here would be example of produced sprite sheet for 2nd vertical and 3rd horizontal rotational level in the system of verRotLevels=3, horRotLevels=12, numberOfFrames=24, numberOfColumns=6, spriteResolution=64. Simulation produces 3x12=36 sprite sheets for this system.



Files, generated by SpritesCreator are written into
/Assets/Resources/3dSprites/modelName/animationName directory.

to animate from single sprite sheet. During the gameplay all rotational levels of requested animation are loaded automatically from
/Assets/Resources/3dSprites/modelName/animationName directory. At the moment it is computationally expensive to use large grids of rotational levels (several hundreds), so user should be carefully. However, user has opportunity to check performance with various sizes of grids before actual usage in the game. SpriteLoader.cs taking parameters from SpritesManagerMaster.cs class, where user should only define “animationName” and “modelName”. There is the screenshot of in game look up for 6x4 example sprites discussed above.

3dSprites also support usage of 3d objects instead of sprites at the distances lower than **“billboardDistance”** set in SpriteLoader.cs for each unit.



3DSprites can use multiple models and animations in the scene (Idle,Walk,Attack and die animations)

Manual control system

“CameraOperator.cs” script, which is attached to Main Camera is the core component of manual control system. This system allows to select player units by click and rectangle. Unit is selected by click if player move mouse over unit and clicks left mouse button (LMB). If LMB is pressed, hold and mouse is dragged is should appear rectangle like in this figure:



Player units, which are inside this transparent rectangle will be selected when LMB is released. Selected units obtain a flag "Is Selected" in "ManualControl.cs" script, which is attached to each units in the same way as "UnitPars.cs". Units can be deselected by click LMB in empty space.

All units which are selected can move around the scene. They starting to move to right mouse button (RMB) click position once RMB is pressed. If RMB is pressed on another unit, all selected units will attack this target unit. If target unit is enemy unit, warning will appear about the danger of war and attack will not be performed. Additional click on the same unit in a few seconds will lead to war and selected units will start to move to destroy target. Manual attack is strict - once it is performed, units will keep following and attacking target until target is destroyed. Player units can also be used as manual targets. In that case no war notice will appear.

Performance

uRTS toolkit has build-in performance reporter for detailed analysis how much computing power is taken by each of 6 phases in BattleSystem.cs script. This reporter is running in game and show performance in right side of the screen. To access performance statistics press “t” in the keyboard.



First integer number show how many objects are used by each phase. Search phase has two more integers, showing how many attackers are searching their targets in each of two teams. First float number show time spend in calculations, second float number show total time spend in calculations and WaitForSeconds and third float shows fraction between these two times in percentage, which is phase performance.

First line represents total number of units used by entire simulator, sums of all 6 phases times respectively, and average performance for all 6 phases.

Selection marks and health bars

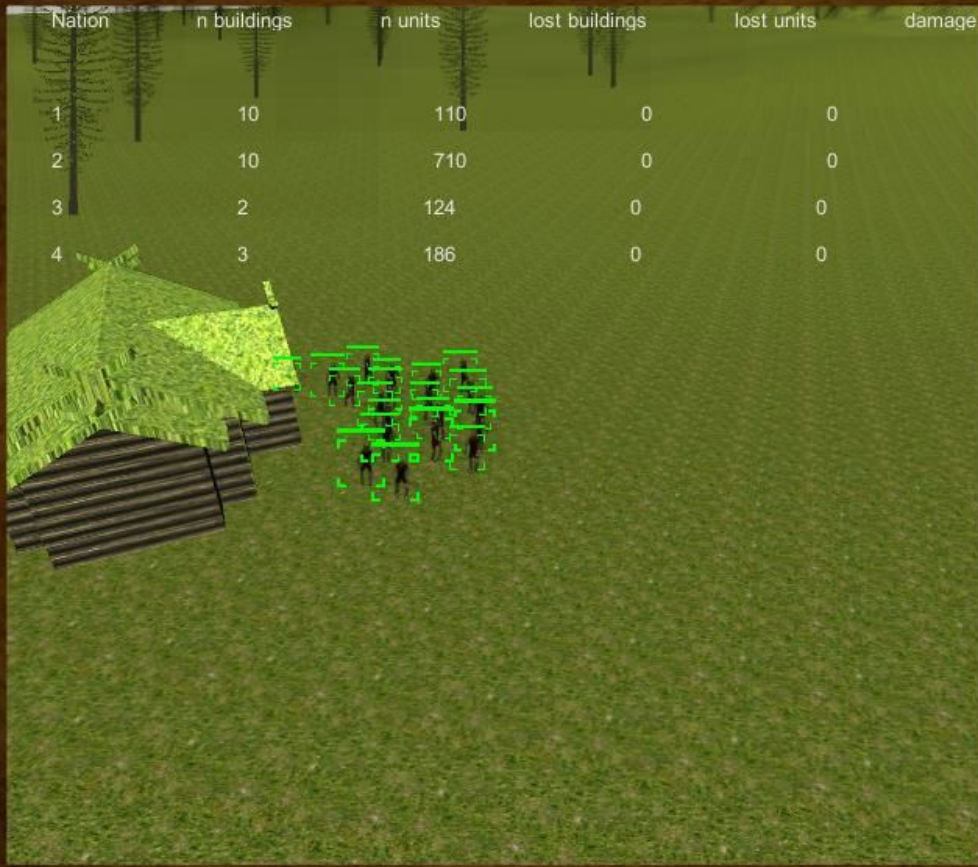
One of graphical components, which allows to see and control players units are health bars and selection marks. They can be shown for selected units.

Selection marks are coloured wedged marks shown around each units like.They are colour coded by unit health status. Green colour denotes that unit is with full health, while red one,unit is almost dead

Health bars are similar features like selection marks, but they indicate how much health each unit has.

Both health bars and selection marks can be switched on and off by pressing "H" in keyboard. Both elements are simple generated textures drawn with OnGUI() function. Note that at the moment showing several thousands health bars or selection marks changes performance quite dramatically.

19250		37500		18600		1856		Options	
Nation	n buildings	n units	lost buildings	lost units	damage made	damage got			
1	10	110	0	0	0	0			
2	10	710	0	0	0	0			
3	2	124	0	0	0	0			
4	3	186	0	0	0	0			



Units grouping and Formations

uRTS supports simple units grouping and formations system. All units, which can move (not buildings) can be selected in amounts larger than one. Selected units can be grouped into a simple group and formation group.

Simple group does not affect how units are moving and only helps for selection purposes.

Formation group is a group of units, which is always affected as formation when units moves.

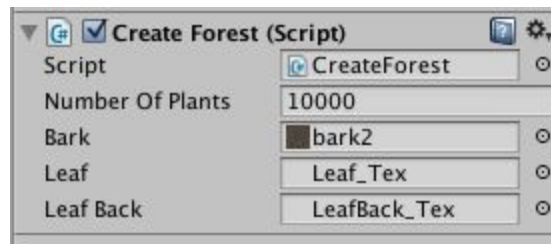
Only all units can be selected in a formation.

Formations are based on formation mask, which can be loaded as an image. Image is a simple white and black pixels collection. When formation of units is being created, units are moving to locations based on where white pixels were seen from texture.

Player can create and destroy formations based on their own choice in game.

Create Forest

CreateForrest.cs is procedurally creating forrest on current scene terrain. Currently script uses single spruce model and user can only modify number of trees placed on the terrain and textures for them.



Search optimisation

uRTS toolkit is allowing to simulate RTS battles with 6 phases: searching targets, approaching targets, attacking targets, damaged units self-healing, dying phase and sinking phase.

Different phases are used only for suitable objects and full script (Battle System.cs) is attached just to one object in entire game (for example terrain), what allows to perform calculations and set parameters for allobjects very quickly.

Nearest neighbour search, used to find nearest targets coordinates is using kdtree.cs class (with additionally used sorting by heapsort algorithm for vectors)

WaitForSeconds used for long loops allows to increase performance even further. In general script is optimised to run with $O(N \log N)$ time assumptions. Performance analysis (which is performed in this script too) show that there are only 3 - 10 % of time spend for calculations inside loops compared with all time with WaitForSeconds.

Each unit has their individual movement, set by script, which is independent from other units and makes battles more realistic.

Trivia

If you experience an error in your code feel free to post on forums or email to chanfort48@gmail.com with the following details explained:

1. Parameters, which you used and how they are different from original ones;
2. Error message;
3. Double click on Error message to get highlighted line in a script: include a few lines of code, where error appeared, that I could look for reason;
4. Any other details, you mentioned and think they could be related with error.

Free scripts adopted

[Required] "KDTree.cs"

<http://forum.unity3d.com/threads/29923-Point-nearest-neighbour-search-class>

SpriteLoader.cs uses SpriteManager script:

<http://wiki.unity3d.com/index.php?title=SpriteManager>

These scripts are available for public in given URLs and also included in uRTS toolkit package scripts folder.

Perlin noise functions used for forest generation are based on:

<http://devmag.org.za/2009/04/25/perlin-noise/>
algorithms.