Simulating evolutions of castles
Battles python module usage

Albert Mas

1 Introduction

To run a city evolution simulation, just instantiate a `Run` object and execute it:

```
from Battles.Run import Run

r = Run(playdataxml = "simulation.xml",
        setttingsxml = "settings.xml")
r.Execute()
```

The `playdataxml` parameter is the XML file with the simulation set of parameters. The `settingsxml` parameter is the XML file with the global application parameters, and it is optional. If it is not specified, it uses the default one: `Battles/Utils/settings.xlm`.

There are two play modes: city evolution and battle simulation. The city evolution simulates the city growth and allow to simulate battles. The battle simulation mode only simulates a battle. On the next sections the XML specifications of each file will be explained.

2 Simulation XML: City evolution

The simulation XML file contains the paramters to define a city evolution. It is wrapped by `<Game></Game>` tag, and the elements that must contain are specified in the next table. All parameters are mandatory, except those ones that are marked as optional.

Parameter	**Description**
`Type`	Type of simulation Available values: `CityExpansion`: Standard simulation `Battle`: See Section 3
`TimeRange`	Simulation time range, in centuries. Format example: `[8, 19]`
`Battlefield`	Battlefield data. See Section 2.1
`Castle`	Castle data. See Section 2.2
`CityEvolutions`	City evolution patterns. See Section 2.3
`CityExpansion`	City expansion dates. See Section 2.4
`BattleEvents`	See Section 2.5
`ExpansionCheckings`	Checkin dates. See Section 2.6
`StarFortress`	Starfortreess creation data. See Section 2.7

You must take into account the next issues:

- There are not any restriction on scale or units, althought is recommended to use meters. You must use the same scale/units for all properties.
- Use decimal values for all measures.
- The main view is a 2D plane view of the castle and the battlefield. The view coordinate systen starts at the top left corner window.

2.1 Battlefield

This element defines the battlefield parameters. The battlefield is just the 2D plane where the simulation runs.

Parameter	**Description**
`Bounding`	Battlefield size. The battlefield is a square. The value is the square side length
`CellSize`	Size of battlefield squares subdivision. The battlefield cell contains only one battalion. Therefore, the cell size defines also the battalion maximum size. Recommended value is 10.0.

`Rivers`	Rivers definition. Rivers are defined by a list of polylines with width. Each polyline is defined as follows on the next example:
	`<Trace>`
	`  <Width>30.0</Width>`
	`  <Polyline>`
	`    <Vertex>`
	`      [28.2, 94.7]`
	`    </Vertex>`
	`    ...`
	`    <Vertex>`
	`      [52.6, 92.8]`
	`    </Vertex>`
	`  </Polyline>`
	`</Trace>`

2.2 Castle

This element defines the castle initial shape.

Parameter	**Description**
`Orientation`	Castle front direction. It is used as reference to construct well oriented squared towers. The value is a 2D vector. Format example: `[0.0, -1.0]` (see Figure 1).

Parameter	Description

`OldCity`	Initial castle shape defined by a list of initial inner houses. The curtain wall will be constructed wrapping around these houses.
	Format example for each house:
	`<House>`
	`  [100.0, 250.0]`
	`</House>`
	, where each value is the house 2D center point (see Figure 1).

`Moat`	If is specified, the castle has a wrapping moat. The tag must has the `<HasWater>` tag that defines if moat has water or not.

`Shape`	Initial castle shape defined by a polyline from a list of vertices, where each vertex is defined by `Vertex` tag. Each vertex has the 2D position and the kind of tower. Format example:
	`<Vertex>`
	`  <Point>`
	`    [970.0, 1125.0]`
	`  </Point>`
	`  <TowerType>`
	`    SquaredTower`
	`  </TowerType>`
	`</Vertex>`
	The tower type values are: `SquaredTower`, `RoundedTower` or `Tower` if a random choice is desired (the random choice is a fuzzy selection from current year simulation over the towers timerange table shown in Section 4.1.2. The tower type is optional.

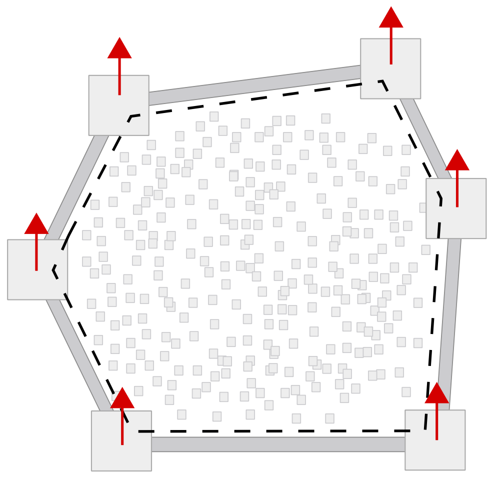

Figure 1: *Castle initial shape from old city bounding and castle oriented squared towers .*

2.3 City Evolutions

This element contains a list of city evolution patterns. Each pattern defines the range time and the main direction of the city growth (in houses). The patterns can intersect in time with other patterns. The evolution patterns are joined in groups wrapped by tag `<Group></Group>`, where each group is defined as follows:

Parameter	Description
`ID`	Group identifier. It is used to be related later by `CityExpansion` (see Section 2.4.
`Evolutions`	List of evolutions, where each evolution is wrapped by `<Evolution></Evolution>` tag.

Each evolution pattern is defined as follows:

Parameter	Description
`TimeRange`	Time range in years. Format example: `[850, 1100]`
`Direction`	Expansion 2D vector. See `SegmentBase` for more information. Format example: `[-0.3, -0.5]`
`HousesPerYear`	Number of houses created by year. Optional parameter (1 by default). You can use values lower than 1 to specify a slow city growth.

`SegmentBase`	Definition of a segment from where the houses start its deployment. The houses deployment starts along the segment and follows the `Direction` vector (see Figure 2. Because the houses expansion are calculated using a cosine distribution, the distribution will be more grazing for a shorter segments, and more sparsed for a longer segments (see Figure 3). This value is optional. If it is not defined, the deployment uses the castle walls as initial segments. The segment base is defined with 2 points. By example:
	`<SegmentBase>`
	`  <P1>[23.1,24.2]</P1>`
	`  <P2>[34.4,25.6]</P2>`
	`</SegmentBase>`

2.4 City Expansion

The city expansion is executed when the castle is wrapped by a new curtain wall. This expansion can be generated by a defeat in a battle or manually using this section data. This section is optional.

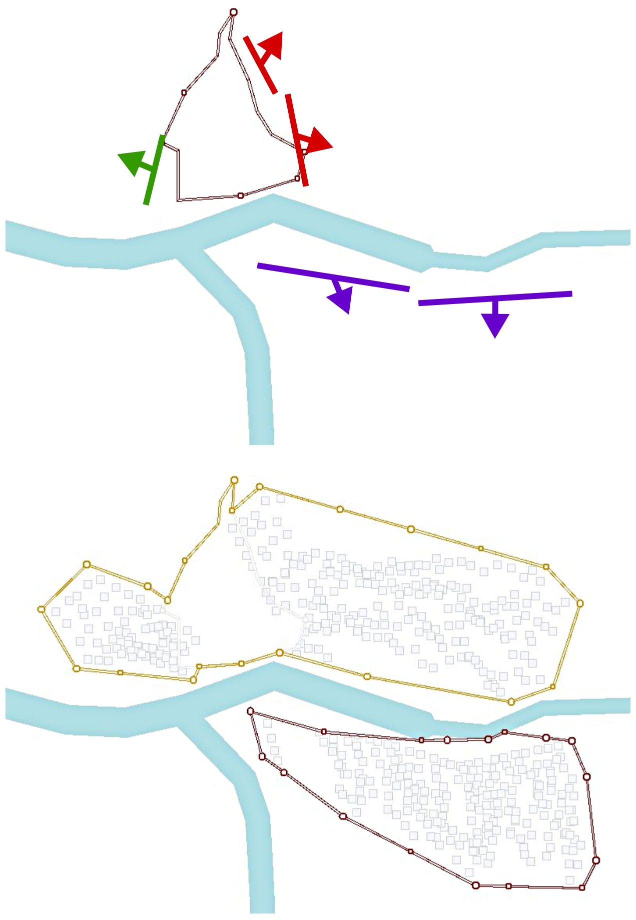

Figure 2: *Example of city expansion using evolution patterns. There are three expansion groups, each one defined by a different arrow color.*

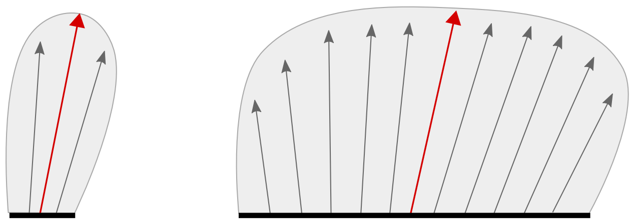

Figure 3: *Houses distribution differences between short and long base segments.*

Parameter	Description
`WallDimensions`	Minimum and maximum wall lengths for all expansions. This parameter is usefull when the user wants to override the default settings value. Format example: `[80.0, 150.0]` , where the values are respectively the minimum and maximum wall lengths.
`Expansion`	Defines the expansion year. Many expansions can be defined, each one related by the tag `<ID></ID>` to a `CastleEvolution` group. Format example: `<Expansion>` `<Year>1384</Year>` `<GroupID>1</GroupID>` `</Expansion>`
`YearsBetween Expansions`	Defines the number of years between city expansions due to battles and manual expansions. If a close battle in time created a new castle, manual expansion is discarded, and viceversa. It is optional

2.5 Battle events

A set of battle simulation can be defined to be executed in a selected year. Each battle event is wrapped by `<Battle></Battle>` tag. Each battle has the next parameters:

Parameter	Description
`Year`	Year of the battle
`Simulations`	Number of simulations. If it is 1, the battle is displayed. Otherwise, it runs in background and shows at the end a list of statistical results.
`RepeatUntil DefendersWin`	If is true, the simulation loops until defender army wins or until the castle cannot be upgraded more.
`Defenders`	Defender army definition. See below.
`Attackers`	Attacker army definition. See below.

The defenders army, wrapped with `<Defenders></Defenders>` tag, is defined by the number of "`Archers`" and "`Cannons`" to deploy. The units are deployed automatically.

Parameter	Description
`Archers`	Number of archers to deploy.
`Cannons`	Number of cannons to deploy. They are deployed on towers before the archers deployment. Then the archers are deployed on the walls and on the empty spaces between cannons on the towers. There must be enough space in towers to deploy the towers (tower width and cell size).

The attackers army are defined as a list of flanks `<Flank></Flank>`. Each flank is a set of battalions that approach to the castle and attack it. The parameters of a flank are:

Parameter	Description
`Direction`	Flank attack direction. If origin is not specified, the flank advances in a directional way. Optional. Format example: `[0.7, 0.7]`
`Origin`	Flank origin position. Optional. Format example: `[100.0, 100.0]`
`StandDistance`	Distance from the castle bounding where the attackers start the battle.

If none origin is specified, the vector is used to intersect a directional ray to the castle bounding circle with increased radius by the stand distance (see case A in Figure 4). Then, the intersection point is the battalion deployment point. If origin is specified but not the stand distance, the troops are deployed on the origin position. If origin and stand distance are specified, the troops are deployed on the intersection point between the castle bounding circle with increased radius by the stand distance, and the ray created with origin and direction (see case B in Figure 4). Finally, if neither direction and origin are specified, a random vector is used in the same way if only direction was specified.

In addition, each flank defines a list of battalions, enclosed by `<Battalions></Battalions>` tag. Each battalion, enclosed by `<Battalion></Battalion>` tag, has the next parameters:

Parameter	Description
`Type`	Type of battalion. The avaliable battalions are “`Infantry`“, ”`Archers`“, ”`Cannons`“ and ”`SiegeTowers`“.
`Number`	Number of battalion units.
`BattalionSize`	Maximum battalion size for each battlefield cell. Optional. If it is not specified, the selected battlefield cells will be full populated. The battalion size is taken into account to populate the cell (see global settings).
`GroupSize`	Grouping size. Allows to deploy groups of battalions with a distance between them defined by GroupsDistance (see Figure 5). This parameter is usefull for deploy sparsed cannons batteries. Optional. 1 by default
`GroupDistance`	Distance between groups. Optional

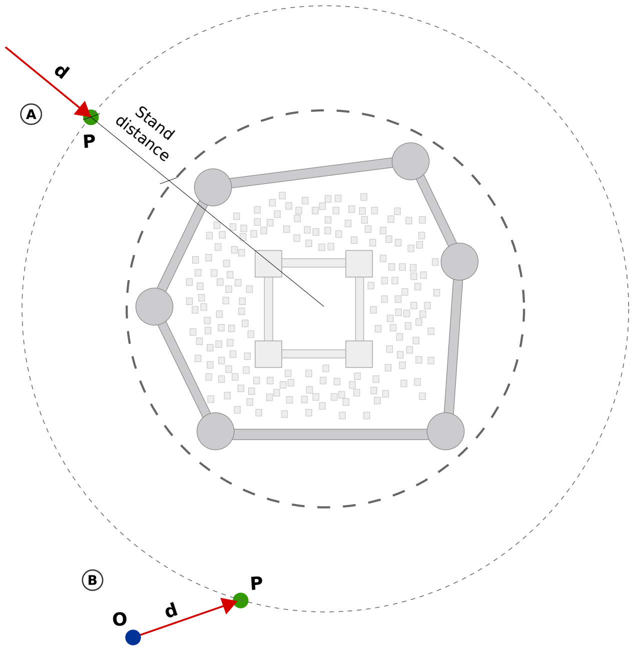

Figure 4: *Flank placement methods. Placement P by only direction vector d (case A), or placement by origin position O and direction vector d (case B).*

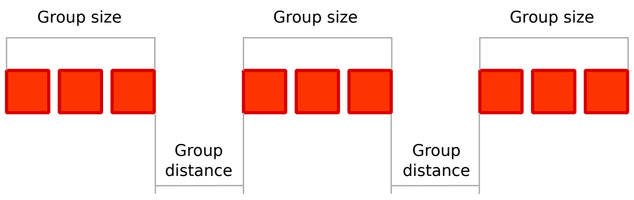

Figure 5: *Battalion grouping.*

Once the deployment starting position is calculated, the battalions deployment around it follow the next rules (see Figure 6):

- The first defined battalion is deployed at stand position. The deployment starts at calculated position and sparses the units along the flank direction perpendicular vector, in both sides.
- The next battalion is deployed just behind the first battalion deployment position. The same for the other battalions. So, the order is important.
- If the deployment of a battalion finds an obstacle (trenches, houses, any castle part or other battalions, jump to the behind row and continues the deployment.
- Battalions of the same kind can be repeated.
- The cannons deployment could be automatically changed if the final stand distance is too close or too far from the castle. See the cannon shoot parameters at global settings.
- The siege towers need a clear and straight path to any castle wall. So, the siege towers could be deployed far from their initial deployment position, searching the best attack position and direction.

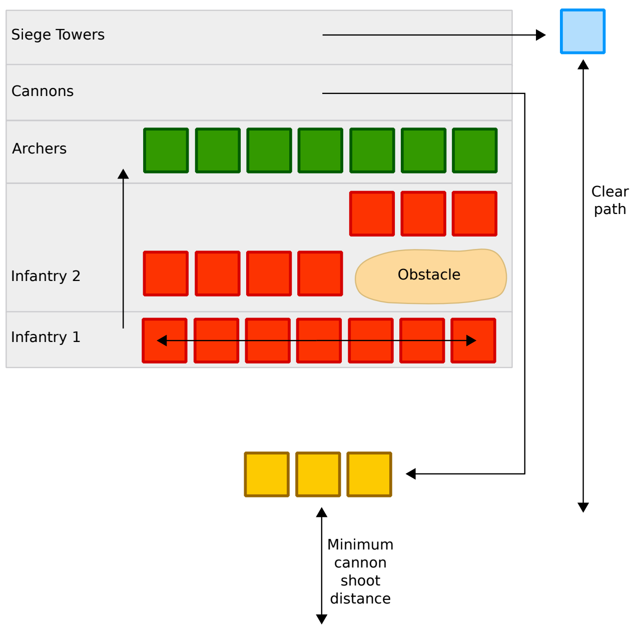

Figure 6: *Battalion deployment on the battlefield.*

2.6 Expansion checkings

This element is a list of years where the expanion history is printed out. It is optional.

Parameter	**Description**
`Year`	Checking year

2.7 Star Fortress

The star fortress is an optional final expansion castle shape. The result is not applicable to any battle, so this shape is calculated at the end of the simulation if it is specified. The next table shows the parameters to construct the star fortress.

Parameter	**Description**
`Activate`	`True` to activate the star fortress shape, or `False` otherwise (or just not define `StarFortress` section.

See Section 4.1.5 to know how to change the default parameters.

3 Simulation XML: Battles

The user can test and check specific battles out of the city evolution context. To do it the `Type` parameter must be defined as `Battle`. The XML structure is different from the city evolution mode, so both parameters cannot be mixed. This simulation mode is just for checking and debuging purposes, and it does not calculate weakest points in the castle structure. The parameters for this play mode are shown on the next table. All of them are mandatory except those that are indicated as optional.

Parameter	**Description**
`Period`	Century of the simulation. Used to select the kind of towers. See Section 4.1.2.
`Battlefield`	Battlefield definition. See Section 3.1.
`Castle`	Castle definition. See section 3.2.
`AttackersPopulate`	Attackers army definition and deployment. See Section 3.3.
`DefendersPopulate`	Defenders army definition. See Section 3.4.
`HasHeightViews`	`True` if the wall height views have to be shown.

3.1 Battlefield

This element is defined as seen in Section 2.1. In addition, it includes in this mode the trenches manual definition. If it is not defined, the trenches are created automatically (see global settings).

Parameter	**Description**
`Bounding`	Battlefield size
`CellSize`	Battlefield cell size
`Trenches`	List of lists of cell coordinates by their indices. Each list is wrapped by a `<Set></Set>` tag. Format example: `<Set>[[20,7], [21,7], [22, 7], [23, 7]</Set>` `<Set>[[8, 8], [8, 9]]</Set>`

3.2 Castle

This element is also defined as seen in Section 2.2. In addition, it includes in this mode the moat activation. If it is not specified the moat is deactivated by default. It is optional.

Parameter	**Description**
`Orientation`	Castle orientation 2D vector.
`OldCity`	List of `<House></House>` with the inner castle houses.
`Moat`	Set the value of `<HasWater></HasWater>` to `True` or `False` to activate the water covered moat or not.

Check the default values in global settings to define the moat parameters such are depth or width.

3.3 Attackers army

The attacker army deployment differs from the city evolution mode. The parameters are defined on the next table:

Parameter	Description
`ArmySize`	List with all kinds of attacker army battalions and their sizes. Available battalions are ”`Infantry`“, ”`Archers`“, ”`Cannons`“ and ”`SiegeTowers`“. Format example: `<Battalion>` `[”Infantry“, 7000]` `</Battalion>`
`Battalion`	See below

Each battalion is deployed by a list of elements wrapped by `<Battalion></Battalion>`. The parameters are shown on the next table The placement is manual, defined by regions. The user can specify cell ranges where to deploy battalions of the same kind. An infantry battalion should be specified as minimum, so the simulation ends when the defenders army defeats or when the attacker army infantry defeats, that is any infantry unit can climb any castle wall.

Parameter	Description
`Type`	Type of battalion. The available types are ”`Infantry`“, ”`Archers`“, ”`Cannons`“ and ”`SiegeTowers`“.
`DeploymentByRange`	Range of cells where to deploy the battalions. The cells are specified by their indices. Format example: `<First>` `[3, 3]` `</First>` `<Last>` `[3, 46]` `</Last>` This deployment overrides the standard deployment method seen in Figure 6.
`BattalionSize`	Battalion size for each battlefield cell. Optional. -1 by default (populate the whole cell)
`SiegeTowerNumber`	Number of siege towers to deploy. The siege towers are deployed automatically, choosing front positions to the weakest walls. This property is only avaiable for siege towers deployments. If it is used, `DeploymentByRange` and `BattalionSize` are not considered. A siege tower needs one infantry battalion to use it as a turtle battalion, that will cover the moat.

3.4 Defenders army

The defenders are deployed automatically on the castle walls and towers. There are only two kind of battalions to deploy on a castle: ”`Archers`“ and ”`Cannons`”

Parameter	Description
`Archers`	Number of archers to deploy.
`Cannons`	Number of cannons to deploy. They are deployed on towers before the archers deployment. Then the archers are deployed on the walls and on the empty spaces between cannons on the towers. There must be enough space in towers to deploy the towers (tower width and cell size).

4 Settings XML

If none `settingsxml` parameter is specified in `Run` class constructor, the default XML settings file is used. It is located at `Battles/Utils/settings.xml`. It is recommended to copy and edit it if some changes are needed.

The structure is wrapped on `<Battles></Battles>` tag. The parameters and elements are shown on the next table:

Parameter	Description
`Castle`	Castle and constructions defaults. See Section 4.1.
`Battlefield`	Battlefield size and trenches. See Section 4.2
`Army`	Armies defaults, for defenders and attackers. See Section 4.3
`City`	City evolution defaults. See Section 4.4
`Game`	Some game miscellaneous parameters. See Section 4.5

4.1 Castle

The castle parameters define the default values for the castle construction elements, such are walls, towers, moat or the star fortress.

Parameter	Description
`Orientation`	Castle front direction. It is used as reference to construct well oriented squared towers. The value is a 2D vector. Format example: `[0.0, -1.0]`
`CurtainWall`	
`OldCityMargin`	Margin around the inner city to construct the initial wrapping curtain wall.
`DefendingLine`	Castle defensive lines settings. It has 3 parameters: `Width`: Wall/tower walkway free space to deploy archers. `CellSize`: Available free space to deploy an unit on the wall. `Height`: Baseline height from the wall/tower walkway.

Parameter	Description
`ShowLabels`	Set to `True` to show each castle construction element label.
`Wall`	See Section 4.1.1.
`Tower`	See Section 4.1.2.
`Bastion`	See Section 4.1.3.
`Moat`	See Section 4.1.4.
`StarFortress`	See Section 4.1.5.

4.1.1 Walls

The walls parameters are shown on the next table.

Parameter	Description
`InnerHeight`	Wall height.
`Thickness`	Wall thickness.
`MerlonHeight`	Merlons height. Used to calculate the walkway heigth. Barely used. At least, it should not be negative or too high.
`WalkwayWidth`	Walkway width. It should be enough big to allow the archer units deployment. Barely used.
`DefenseIncrease`	Increase defenders defense factor due the wall protection.
`BattalionGrid` `CellSize`	Available free space for each deployed unit on the wall. Currently, only archers are allowed to be deployed on walls.
`DefenseAngle`	Angle ranges that permit the wall defense. It has two parameters: `H`: 2D plain view angle (degrees) around the main attack vector (wall normal vector). `V`: Vertical angles (in height), from bottom to main attack vector, and from this vector to top.
`Tile`	Default parameters for the wall tiles. They have four parameters: `Width`: Tile width. `Height`: Tile height. `Resistance`: Tile resistance. The value, applied to each tile, is decreased as it receives cannon ball impacts. `RubbleConversionFactor`: When a cannon ball impacto to a tile, a part of the tile material is converted to rubble that falls to the floor. The parameter value is the percentages of rubble that fall at tile left, front and right sides (see Figure 7). Format example: `[0.25, 0.5, 0.25]`

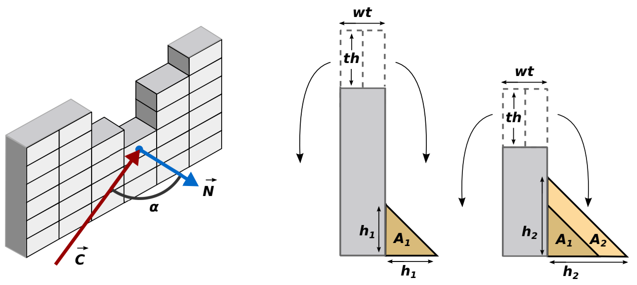

Figure 7: *Wall breaking by cannon shoots on wall tiles (left). When a tile is broken, its material falls down creating a slope (center and right), where wt is the wall thickness and th is the tile height.*

4.1.2 Towers

There are three kind of towers: squared, rounded and bastions. The tower type selection is done automatically considering the century of creation. Althought the bastion is a kind of tower due it is also a construction element that join walls, it is explained at the next section due it has a different set of parameters. In addition, some of the next values are used too for bastions.

The towers parameters are shown on the next table:

Parameter	Description
`TimeRange`	Ranges of centuries to choose the type of tower to construct. The ranges should overlap, so the system uses a fuzzy algorithm between them. There are three time ranges: `Squared`, `Rounded` and `Bastion`.
`InnerHeight`	Tower height.
`Thickness`	Tower thickness. This value should be enough big to allow the cannons deployment.
`SquareSide`	Length of squared tower side.
`CircleRadius`	Radius of rounded tower.
`DefenseIncrease`	Increase defenders defense factor due the tower protection. Used also for bastions.
`BattalionGrid` `CellSize`	Available free space for each deployed unit on the tower. Due the towers allow the cannons and archers deployment, it has two values: `Large`: Space to deploy a cannon. `Small`: Space to deploy an archer.

Parameter	Description
`DefenseAngle`	Angle ranges that permit the tower defense. Used also for bastions. It has two parameters: `H`: 2D plain view angle (degrees) around the main attack vector. The main vector for rounded towers is the radial vector. For the squared ones and bastions, the main vector depends on the side, that is each side normal vector. `V`: Vertical angles (in height), from bottom to main attack vector, and from this vector to top.
`RequiredDistance NeighborFactor`	Value multiplied by the required free distance around a tower to place another one. It is recommended to use values greater than 2. It is optional. The default value is 3.

4.1.3 Bastion

The bastion settings are shown on the next table. Bastions also use some tower settings (see last section).

Parameter	Description
`VirtualCircle Radius`	Radius of the virtual rounded tower used to create the bastion. (see Figure 8)
`Thickness`	Bastion wall thickness. The relation between the circle radius and thickness is important. The radius cannot be less than thickness. If both are close, the bastion will be almost closed. In this case, the system will clamp the radius to the thickness.
`Height`	Bastion height.
`BattalionGrid CellSize`	Available free space for each deployed unit on the bastion. Due the bastions allow the cannons and archers deployment, it has two values: `Large`: Space to deploy a cannon. `Small`: Space to deploy an archer.
`MinDistance`	Minimum distance between bastions. Used when the towers are transformed into bastions to construct the starfortress. If there is not enough free space to fit a new bastion, its size is decreased, or it is removed.

4.1.4 Moat

The moat is automatically created around the curtain wall and towers. The parameters that define it are:

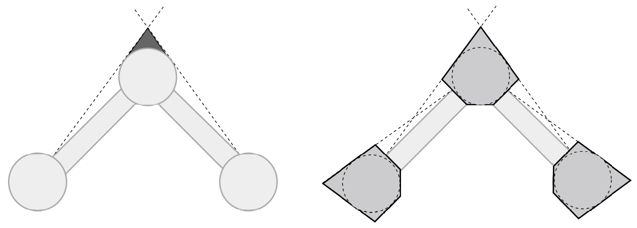
Figure 8: *Bastion creation from rounded towers to cover blind spots.*

Parameter	Description
`Depth`	Moat depth.
`Width`	Moat width, or thickness in 2D plan view.
`HasWater`	Set to `True` if moat has water.
`PenaltyWater`	Penalty movement applied to battalions movement on the battlefield for watered moats. The value is a percentage that is multiplied by the itself battalion movement penalties. The value 1.0 means no penalty.
`PenaltyNoWater`	Like `PenaltyWater`, but applied on moats without water.

4.1.5 Star Fortress

The next table shows the parameters to construct the star fortress. Some of star fortress elements are optional, such are half moons (*lunettes*) and covert way.

Parameter	Description
`Ravelin`	The ravelins construcions have two parameters (see Figure 9): `Method`: Ravelin construction method. Values can be 1 or 2: 1: Ravelin flanks constructed at `<BastionAngle>` from bastion flanks. 2: Ravelin frontal length constructed with `<Radius>` distance. *NOTE: Currently the method 1 is deprecated and should be checked before use it. Only for symmetric fortresses.* `BastionAngle`: Angle between bastion flanks and ravelin flanks. It shouldnt be less than 90 degrees. `Radius`: Circle radius to construct ravelins (like bastions). `MinimumWidth`: Minimum ravelin width. Used to avoid too thin ravelins.
`HalfMoon`	The halfmoons (or *lunettes*) are optional. Their parameters are (see Figure 10): `Activate`: `True` to activate them. `CircleOffset`: Distance between halfmoon center and halfmoon circle radius. `Length`: Distance from the halfmoon circle center to the halfmoon jag.

Parameter	Description
`CovertWay`	The covert way is mandatory, so it wraps the castle, ravelins and halfmoons. Its parameters are (see Figure 11): `Thickness`: Thickness of the covert way. `Offset`: Distance from the half-moons/ravelins to the covertway (see distance d on Figure 11). `PlaceOfArms`: `True` if places of arms have to be constructed on each non-convex angle of the covertway external shape. `PlaceOfArmsLength`: Length of the squared places of arms. `MinimumSegmentLength`: Minimum segment length of covert way polyline. Used to avoid wrong polylines with too small segments. `GlacisThickness`: Thickness of the glacis, the outer wrapper to the covertway, and the most external star fortress defense.

There are not any bad shape control method, so some parameter combinations could produce wrong shapes. The default values should work well for many castles.

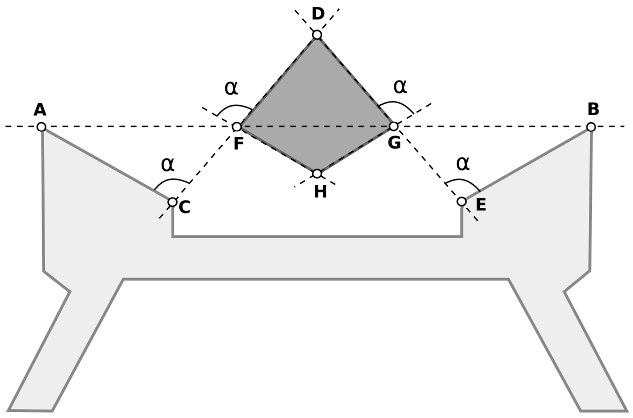

Figure 9: *Star fortress: Ravelins geometry. Points F and G are calculated intersecting the segment $\overline{AB}$ with segments $\overline{CD}$ and $\overline{ED}$. The latter ones are the respective bastion flanks $\overline{AC}$ and $\overline{BE}$ at an user defined angle, close to perpendicular. Finally, from points F and G, two segments, parallel to the bastion flanks, are intersected to get the point H.*

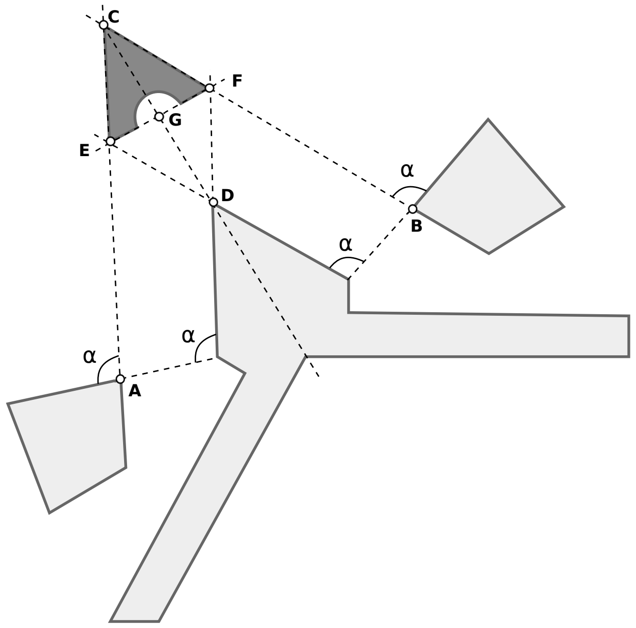

Figure 10: *Star fortress: Halfmoon (or Lunettes) geometry. From the rear ravelin flanks, two vectors are traced parallel to the bastion flanks to get the point C. Then, the bastion flanks are extended and intersected with the segments $\overline{AC}$ and $\overline{BC}$ to get the points E and G, used both to construct the rear lunette flank. Finally, the bastion axis is intersected with segment $\overline{EG}$ and a half circle with an user defined radius is constructed on the back.*

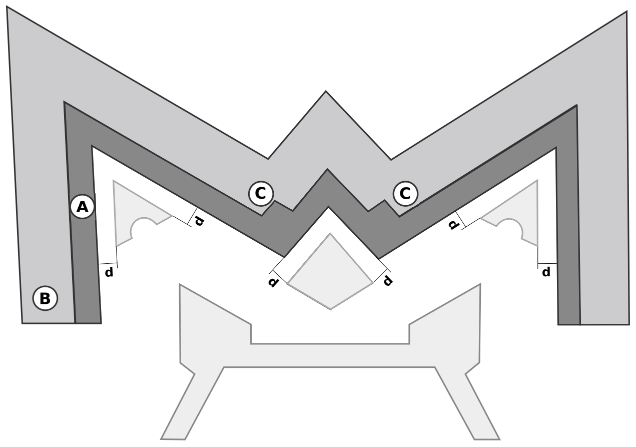

Figure 11: *Star fortress: Covert way, composed by the curtain wall (A), the glacis (B) and the places of arms (C).*

4.2 Battlefield

The battlefield section defines the 2D plan view battlefield sizes and the trenches settings. The trenches are created automatically and randomly. The battlefield must be a square.

Parameter	Description
`size`	Battlefield side length
`GroundCell`	Battlefield cell definition. Many of its parameters are applied to all cells. The API allows to define other values for specific cells. The parameters are: `Size`: Cell side length. `Height`: Cells height. Currently not used. `DefenseIncrease`: Defense value to add at troops deployed on any cell. `MovementPenalty`: Penalty movement applied to battalion movement on the battlefield. It is a percentage that is multiplied by the battalion speed factor. 1.0 means no penalty.
`Trench`	Trenches parameters. There are six parameters: `DefenseIncrease`: Defense value to add at troops deployed on any trench. `MovementPenalty`: Penalty movement for troops on any trench. `ShowOutline`: Set to `True` if trench outline display is required. For huge battlefields is recommended `False` value. `RandomDeployment`: Percentage of trenches density along all battlefield. 1 means all cells become trenches. `RandomDeploymentConsecutive`: Trenches size percentage. It defines the probability to get a trench sequence. `RandomDeploymentMaxTries`: Maximum number of tries to get a consecutive trench. Used basically to avoid infinite loops if *RandomDeploymentConsecutive* is closer to 1.
`River`	Defines the river settings: `PenaltyMovement`: Penalty movement for units through a river (like moats penalty).

4.3 Armies

This section covers the settings for both armies, defenders and attackers. There are specific settings for each kind of battalion: Infantry, Archers, Cannons, Siege Towers and Throwers (see related sections). For each kind of battalion, some parameters are common, such are defense or attack values. The values are not scaled or in any metric, and must be related between the different kind of battalions. Also, these parameters must be related with the other game elements, such are battlefield (speed, by example) and castle construction elements (tile resistance, by example). Some of these values are not used in some kind of battalion, but they are specified just to keep the format and standard battle unit definition.

Parameter	Description
`ShowLabels`	Set to `True` to display the battalions labels. Usefull for debug purposes.
`ShowOutline`	Set to `False` to hide the battalion outline. It is recommended for huge battlefields, where the outline color could overlap the battalion color.
`HumanFieldOfView`	Field of view used for aiming an objective. Used by archers and cannons.
`Infantry`	See Section 4.3.1.
`Archers`	See Section 4.3.2.
`Cannons`	See Section 4.3.3.
`SiegeTowers`	See Section 4.3.4.
`Throwers`	See Section 4.3.5.

4.3.1 Infantry

The infantry only plays on the battlefield for the attackers army. Their goal is to advance to the castle and to climb any wall. Therefore, the attack parameters are not considered.

Parameter	Description
`Defense`	Defense value, compared to the attack value. Should be low.
`Attack`	Attack value, compared to the defense value. Infantry units do not attack, so the parameter is not considered.
`Speed`	Movement speed, in distance per turn/step. By example, if battlefield cell size is 10 and speed is 10, the infantry battalions will move to the next cell each turn. *NOTE: Currently a battalion cannot move faster than one cell per turn.*
`Reload`	Reload time. Not used.
`Accuracy`	Accuracy shoting. Not used.
`Distance`	Shooting maximum distance. Not used.
`Bounding`	Soldier bounding size. Used to know how many soldiers can fit into a battalion that is deployed on a cell. The parameters are `Length`, `Width` and `Height`.
`ClimbSpeed`	Climbing speed.
`Stationary`	`True` if unit cannot move. For infantry it must be `False`.
`MovementPriority`	Prority value to decide what unit to move when two of them collide into the same battlefield cell. A battalion with higer value will swap current battalion position by its position. Minimum value: 0.

`MovementPriority WaitingClimbing`	This priority value overrides `MovementPriority` when the battalion is climbing. The value should be high, so nobody can move the battalion when it is climbing, even siege towers.
`RubbleClimbSpeed`	Movement speed when infantry moves over wall rubble.
`SearchRadius GoToRumble`	When the infantry is in front of a wall, it decides to climb it on the nearest position or search for any accumulated rubble. This value is the radius search.

4.3.2 Archers

The archers are the representation of the soldiers who can shoot, either arrows or bullets. The archers can be deployed on the battlefield as attackers, or on the castle as defenders. The battlefield archers can move to reach a good aiming position, and have not any aiming angle restriction. The castle archers do not move of their position, and inherits the aiming angle restrictions from the castle element where they are deployed. Therefore, the movement parameters are only applied to the battlefield archers.

Parameter	**Description**
`Defense`	Defense value, compared to the attack value. Should be low.
`Attack`	Attack value, compared to the defense value.
`Speed`	Movement speed, in distance per turn/step.
`Reload`	Reload time. Number of turns/steps to wait until the archer can shoot again.
`Accuracy`	Accuracy shoot factor, in range [0-100]% (see Figure 12).
`Distance`	Shooting maximum distance (see Figure 12).
`Bounding`	Soldier bounding size. Used to know how many soldiers can fit into a battalion that is deployed on a cell. The parameters are `Length`, `Width` and `Height`.
`Stationary`	`True` if unit cannot move. Archers are stationary by nature (except when they move to reach a good aiming position). It must be `True` in any case.
`MovementPriority`	Prority value to decide what unit to move when two of them collide into the same battlefield cell. A battalion with higer value will swap current battalion position by its position. Minimum value: 0.
`DefenseShoot DoubleCheck`	See below.
`ShootsToStay`	Percentage [0,1]% of shoots in attack range performed by an archers battalion to decide to stay in its place or to move to a better one. 0 means an absolutely stopped archer, althought it will cannot aim to any target. This parameter is usefull to control the relationship between the archer goal about be covered by a trench, and the archer goal to shoot the closest target. Applied only to the battlefield archers.
`SearchRadius Trench`	Search distance to looking for a trench where to be covered from castle shoots. Applied only to the battlefield archers.
`Defenders MarginSpace`	Minimum distance between each defender archer in walls or towers. Optional. 0 by default.

When an archer or cannon search for the best target to shoot, it uses the battlefield center position as a fast reference. On of the considered parameters is the visibility to the target. After the cell with the target is selected, a random position inside cell is choosen. If `DefenseShootDoubleCheck` parameter is `True`, the algorithm checks again for the visibility to the new point. The visibility checking is computionally expensive. If the battlefield cells are enough small, it would be good enough setting this flag to `False`.

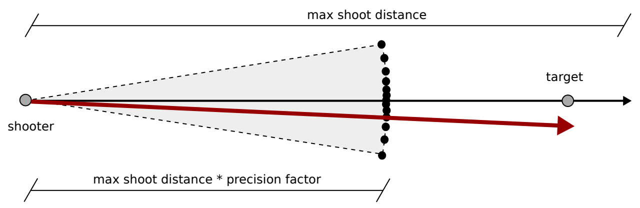

Figure 12: *Shooting method using the precision factor over the maximum shoot distance.*

4.3.3 Cannons

The cannons are the representation of the artillery. They cannot move, so the movement parameters are not used. The soldiers who control the artillery are not considered.

Parameter	Description
`Defense`	Defense value, compared to the attack value. The cannons should have a high defense value.
`Attack`	Attack value, compared to the defense value.
`Speed`	Movement speed. It should be 0.
`Reload`	Reload time. Number of turns/steps to wait until the cannon can shoot again.
`Accuracy`	Accuracy shoot factor, in range [0-100]%
`Distance`	Shooting maximum distance.
`Bounding`	Cannon bounding size. Usually a cannon is enough big to fit only one in one cell. Be aware to set a size less than battalion cell or castle cell sizes. The parameters are `Length`, `Width` and `Height`.
`Stationary`	A cannon cannot move. It must be `True`.
`MovementPriority`	Nothing can move a cannon, so the parameter value should be the larger one.
`DefenseShoot` `DoubleCheck`	See Section 4.3.2.
`ShootAngle`	Angular shooting bounds. The parameters are: `H`: 2D plain view angle (degrees) around the main attack vector. The main vector is the vector from the cannon and target positions. `V`: Vertical angles (in height), from bottom to main attack vector, and from this vector to top.
`BallRadius`	Approximated cannon ball radius. It is used when a castle cannon shoots against a battlefield battalion. The ball size is used to estimate the number of killed soldiers.
`DefaultPlacement` `Distance`	Percentage [0,1]% of the shoot distance used as guide to place the cannons from walls. Low values means best shoots (closer to walls), but less defense condition.

4.3.4 Siege towers

The siege tower action starts by its construction. At the same time, an infantry unit is selected to cover the castle moat and allow the advance of the structure. This infantry unit is named `Turtle`, it can receive shoots from the castle, like another battalion, and it is protected with an effective defense method. The siege tower has different levels. In each level there are archers deployed automatically. The number of levels depend on the height of the castle walls, and are selected automatically.

Parameter	Description
`Defense`	Defense value.
`Attack`	Attack value. It shold be the same value than archers attack parameter. The algorithm multiplies it by the number of internal archers, that is unknown until the siege tower is constructed.
`Speed`	Movement speed. It should be slow.
`Reload`	Reload time. It should be the same thant the archers reload time parameter.
`Accuracy`	Accuracy shoot factor, in range [0-100]%. It should be the same thant the archers accuracy parameter.
`Distance`	Shooting maximum distance. It should be the same thant the archers distance parameter
`Bounding`	Bounding size. The parameters are `Length`, `Width` and `Height`. The height parameter is not considered, so it is unknonw until the siege tower is constructed.
`Stationary`	It must be `False`.
`Movement` `Priority`	It should has more priority than archers or infantry, but less than cannons. Therefore, a cannon cannot be on the siege tower path to the selected castle wall.
`LevelHeight`	Height for each siege tower level. It should be enough to fit the archers height. The number of siege tower levels is calculated automatically from this height and the castle wall height.
`Construction` `TimePerLevel`	Time spent for the construction of each siege tower level.
`TurtleDefense`	Increased defense to the turtle battalion.
`CoverMoatSpeed`	Spent time to cover one moat cell by the turtle. It should be less than 1.

4.3.5 Throwers

The throwers are special units created from the closer archers on the castle walls when the attackers start to climb. The throwers throw dissuasive material to the climbers. When there are no more climbers, the throwers are dissolved, becoming again normal archers.

Parameter	**Description**
`Defense`	Defense value. It should be the same than archers.
`Attack`	Attack value. The damage is applied by distance. The applied damage is the attack divided by the distance per throwers `Distance` parameter value.
`Speed`	Movement speed. The throwers do not move.
`Reload`	Reload time. This value will be divided by the battalion size. So, more units on the battalion, less reload time.
`Accuracy`	Accuracy shoot factor, in range [0-100]%. It should be high, so it is easy to shoot a set of soldiers climbing in height.
`Distance`	Distance attenuation (see `Attack` value). Greater value means more attenuation.
`Bounding`	Bounding size. The parameters are `Length`, `Width` and `Height`. The height should be the same than the archers. The length and width are not used, so they are calculated when the thrower battalion is created.
`Stationary`	It must be `True`.
`Movement Priority`	Not used.
`Batttalion MaxSize`	Maximum number of archers in the battalion. If there are not any closer archer, the battalion is created with less archers.

4.4 City Evolution

The next table shows the parameters to control the city evolution simulation mode. See Section 2.3 for the other non-global parameters.

Parameter	**Description**
`EvolutionSpeed`	Default simulation speed. Minimum 1, the faster mode.
`YearsPerStep`	Spent years for each turn/step.
`MinWallLength`	Minimum wall length when a new curtain wall is created.
`MaxWallLength`	Maximum wall length when a new curtain wall is created.
`MatchVertices Distance`	Margin distance to match vertices between two castles in an union operation. Small values means close vertices, therefore close towers. Great values means less close vertices, but some walls could disappear.
`WaitBattle`	This is the time that the system has to spend before check if a battle has finished. This is an issue related with the GUI and threads system, where a manual checking on the thread ending is mandatory. Small values mean many checkings, but large values mean the system could spent too much after a battle end. A value between 1 and 200 should be enough, althought it depends on the used hardware.
`DisplayOldTownGrid`	Displays the inner castle grid. Only for debug purposes.
`Houses`	See below.

Each house is represented as a small square deployed outside the castle curtain wall. The deployment is automatic, using a cosinus random distribution to get a simple urban evolution simulation. The houses can intersect between them. The houses creation are controlled by next parameters:

Parameter	**Description**
`Size`	Side length of the square that represents a house.
`DistanceWall`	Distance from the curtain wall to deploy the first houses.
`MinDistanceBetween`	Minimum distance between houses. This is only an orientative value, so the minimum distance cannot be controlled due the random deployment (none rejection sampling is used due performance reasons). Try different values and combine them with the maximum distance to get different results.
`MaxDistanceBetween`	Maximum distance between houses.
`PlacementFuzzy`	Factor value to apply some jittering to the houses deployment to avoid the patterns that can produce the algorithm. Minimum 1. Larger values means wider and longer radial spaces between houses (them could be considered as some kind of streets).

Parameter	**Description**
`CreationPerYear`	Houses created each year.
`PreferenceFactor`	Priority percentage [0,1]% on the selection of the older houses to construct new ones aside. Large values could produce an elongate city shape.

4.5 Game

On the next table there are some paramaters related to the game display:

`speed`	Game speed. Minimum 1 (faster).
`WindowHeight`	Window height. Due UI issues, windows and viewports have to be squared.
`WindowWidth`	Window width.
`ViewportHeight`	Viewport height.
`ViewportWidth`	Viewport width.
`HeightViewHeight`	Height of wall height views.
`HeightViewWidth`	Width of wall width views.
`ShowGrid`	Displays a battlefield grid. Not recommended for huge battlefields.