

Platform for massive multiplayer programming games

Héctor Ramón Jiménez

Facultat d'Informàtica de Barcelona

January 28, 2016

Overview

1 Introduction

2 Formulation

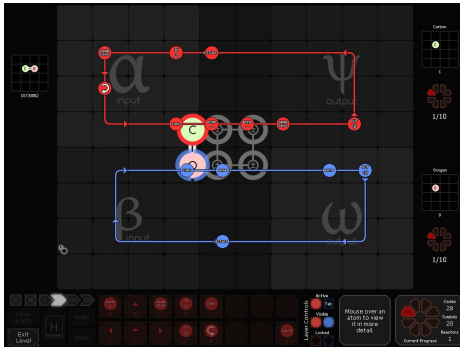
3 Implementation

4 Demonstration

5 Evaluation

Programming games

- Players do not directly interact with the game.
- Players write computer programs (AI) that play the game.



SpaceChem

Programming games can:

- show how algorithms work in a visual way.
- motivate players to learn and improve.

Programming games can:

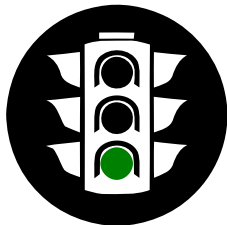
- show how algorithms work in a visual way.
- motivate players to learn and improve.



They are commonly used in education to teach students different programming techniques.

The EDA competition

An AI programming challenge held every semester at the FIB using Jutge.org



The problem

Ronda 28

Torn 1

000072374	EasyGG_v5c	10000	UltimateCR4	5909	Pertur_LXIX	17974	Wasabi_v01c	6000
Pendent	MigoKebab	?	SantaSativa3	?	JanBrown5	?	Mitsukurina2	?
Pendent	MrTrivi4	?	Grimmjow6	?	supernov1kk4	?	Bardock_v11	?
Pendent	Lluis_23	?	deadlock_7	?	Frida_Kahlo	?	BaconQueso4	?
Pendent	l1	?	DemPlayzzz	?	cor_petit	?	yattt_v6	?
Pendent	Satanas	?	collector	?	B_d_D_v4	?	Pilgrim	?
Pendent	Furlas3	?	Mazik5	?	Ark	?	TERMINATOR7	?

Jutge.org round system

The problem



Twitter bot announcing eliminated players

The state of the art

- Google's AI programming challenge
- Battlecode
- CodinGame
- EDA competition

All of them feature multiplayer programming games with a limited amount of players per match.

The main objective

Develop a set of components that ease the creation and the usage of *massive* multiplayer programming games (MMPGs).

Secondary objectives

- 1 Develop an abstract game engine
- 2 Allow hot-swapping of AIs
- 3 Implement a real-time webviewer
- 4 Make the infrastructure scalable and stable
- 5 Create a game example

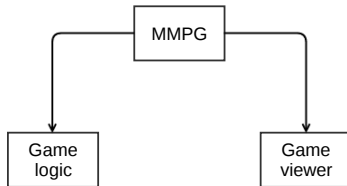
Main benefits

- For students:
 - Continuous learning process
 - More attachment to the match
 - More interesting games
- For professors:
 - Reduced amount of work
 - No need to arrange multiple matches
- For game developers:
 - Easiness to build new MMPGs
 - Freedom to create huge worlds

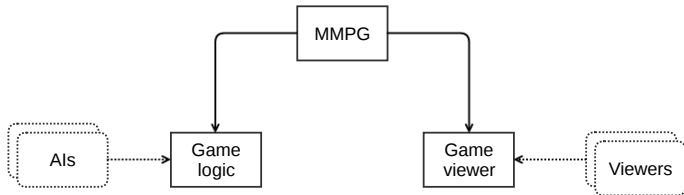
The solution

MMPG

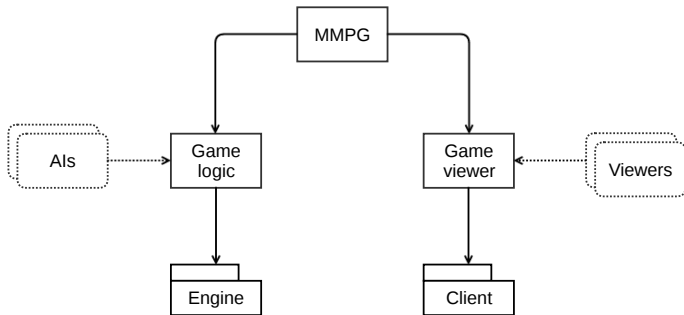
The solution



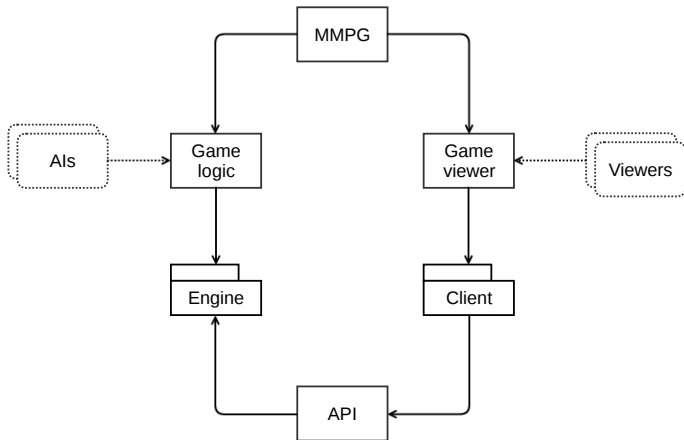
The solution



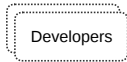
The solution



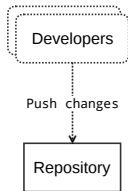
The solution



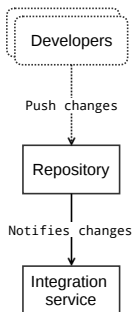
Continuous integration



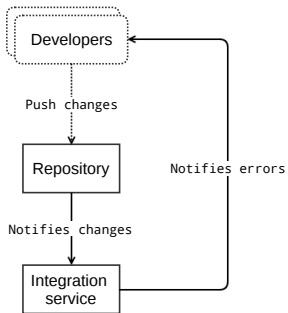
Continuous integration



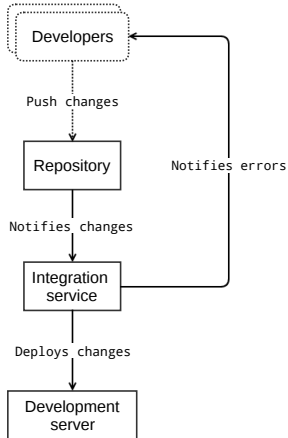
Continuous integration



Continuous integration



Continuous integration



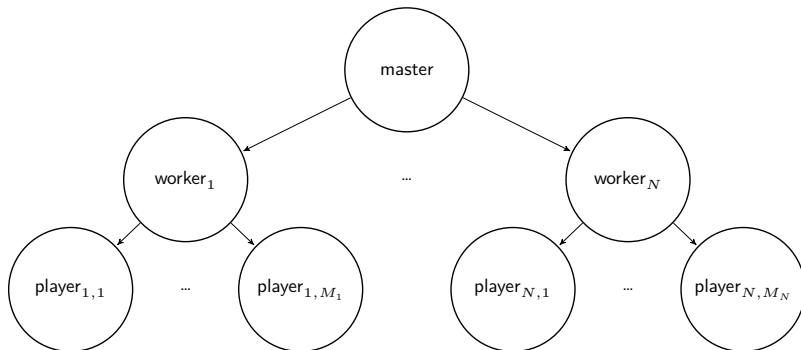
A simple prototype

Engine: Running one simple player and notifying actions to subscribers.

API: Subscribed to the engine and notifying actions to clients.

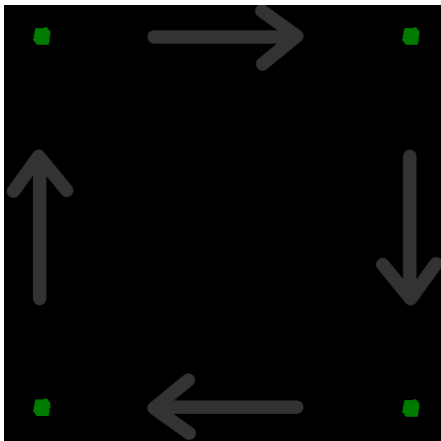
Client: Subscribed to the API and drawing player actions in a web-browser.

The engine architecture



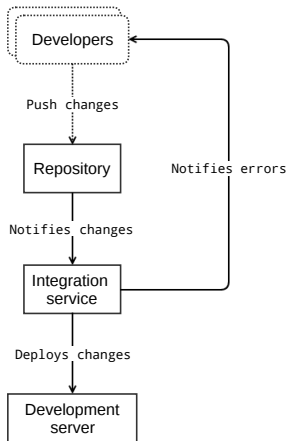
Hierarchy of engine processes with N workers and $\sum_{i=1}^N M_i$ players

The prototype result

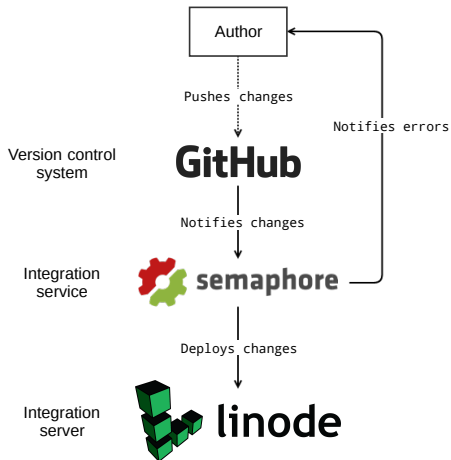


A cube that was moved by the player program

Continuous integration stack



Continuous integration stack



Match replay

An MMPG match might last weeks.

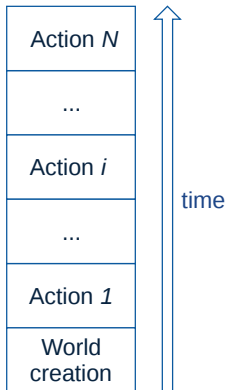


Players will miss parts of the match.



Players need to be able to replay the past of the match.

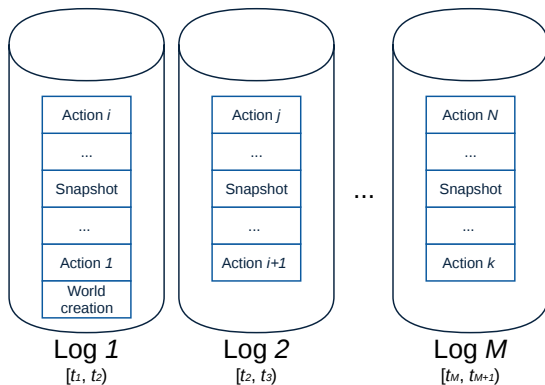
Event sourcing



Snapshots

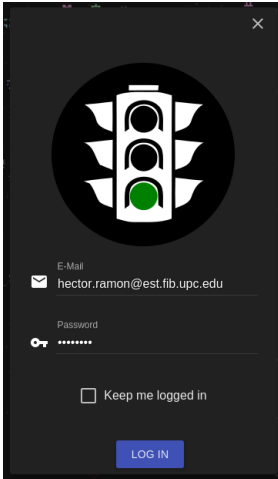


Log files



$$t_2 - t_1 = t_3 - t_2 = \dots = t_{M+1} - t_M = \text{log interval}$$

Authentication

A login form with a dark gray background. At the top center is a circular logo featuring a white traffic light with a green light illuminated. Below the logo, there are two input fields: one for 'E-Mail' with the address 'hector.ramon@est.fib.upc.edu' and one for 'Password' with masked characters. A 'Keep me logged in' checkbox is located below the password field. At the bottom center is a blue 'LOG IN' button. A close button (X) is in the top right corner.

E-Mail
✉ hector.ramon@est.fib.upc.edu

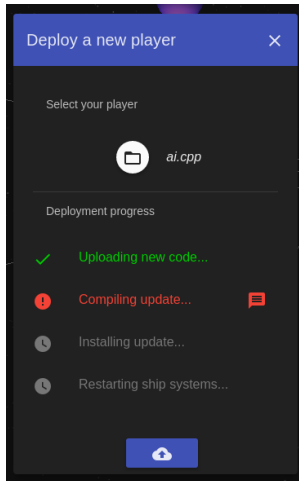
Password
🔑 ••••••

☐ Keep me logged in

LOG IN

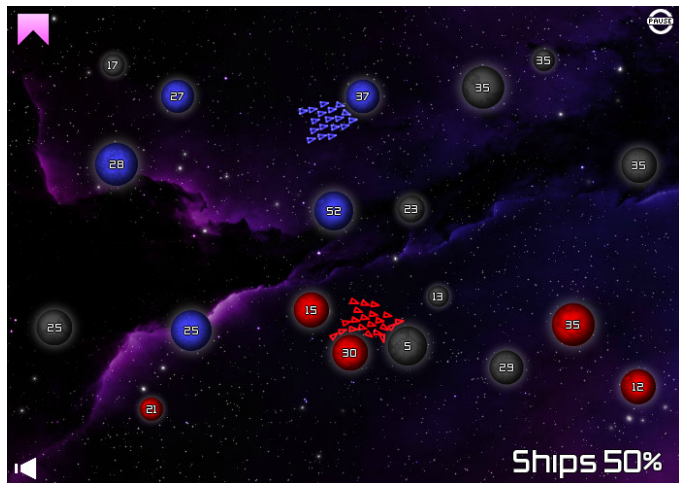
Login form

AI hot-swapping



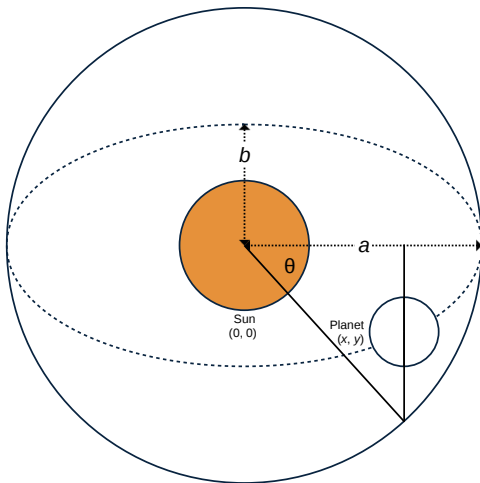
Player deployment

Galcon



Galcon 2

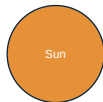
Planetary system approximation



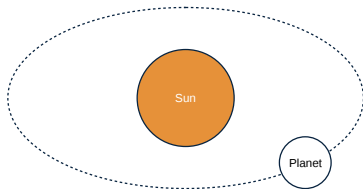
$$x = a \cos\theta$$

$$y = b \sin\theta$$

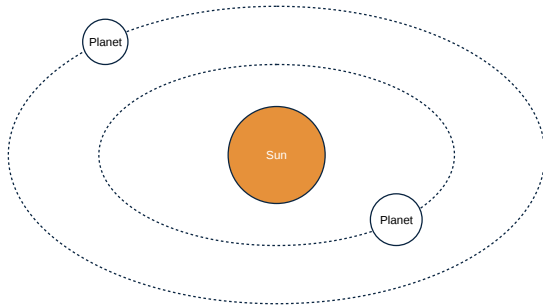
Procedural generation of planetary systems



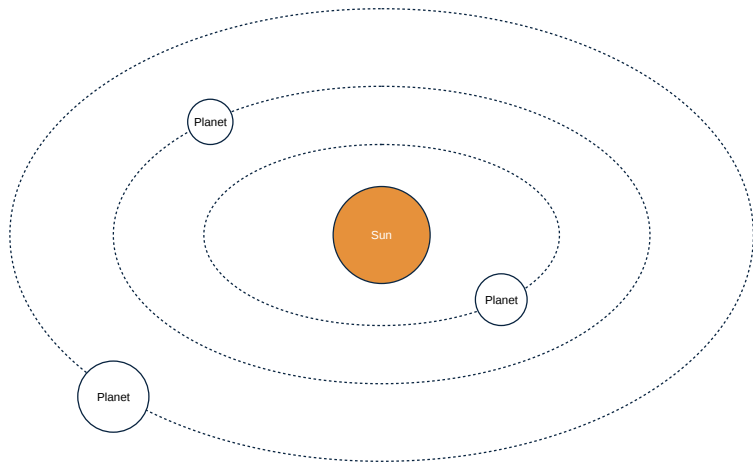
Procedural generation of planetary systems



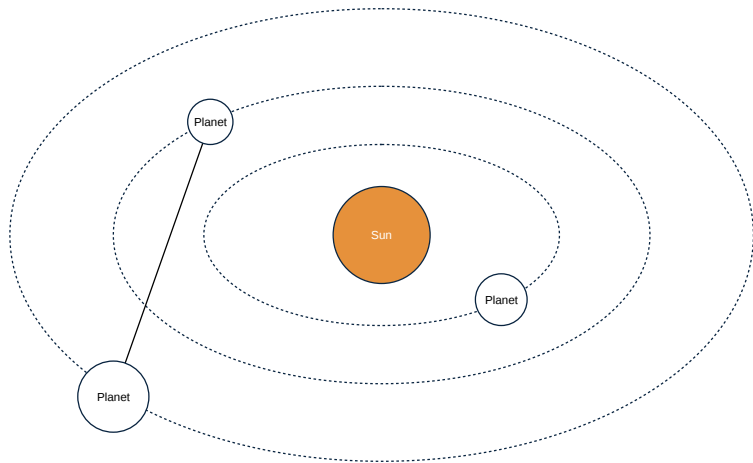
Procedural generation of planetary systems



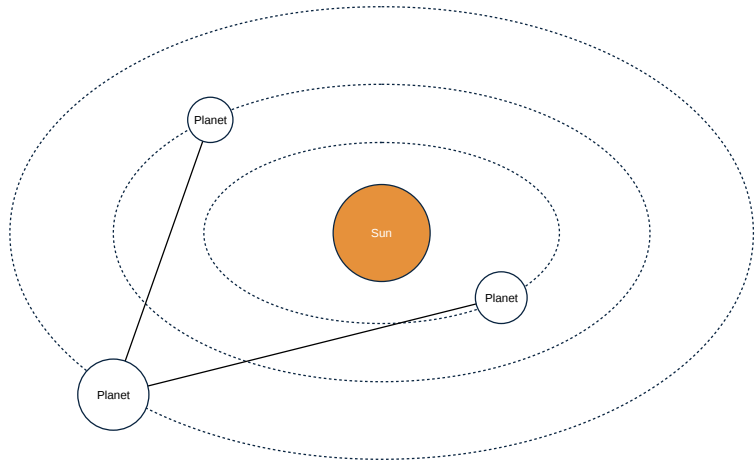
Procedural generation of planetary systems



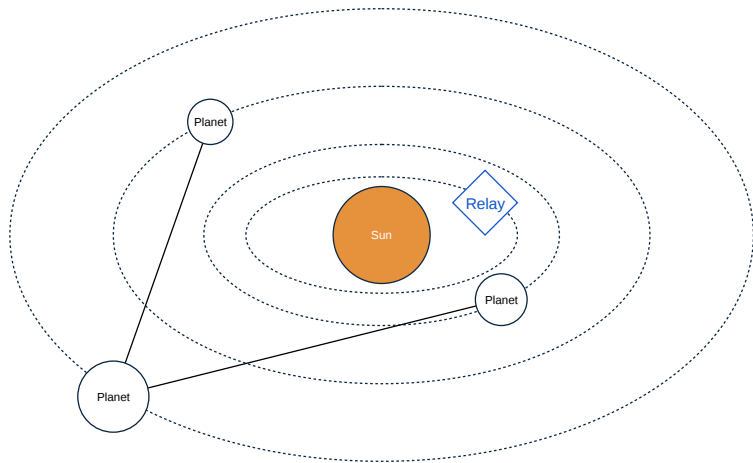
Procedural generation of planetary systems



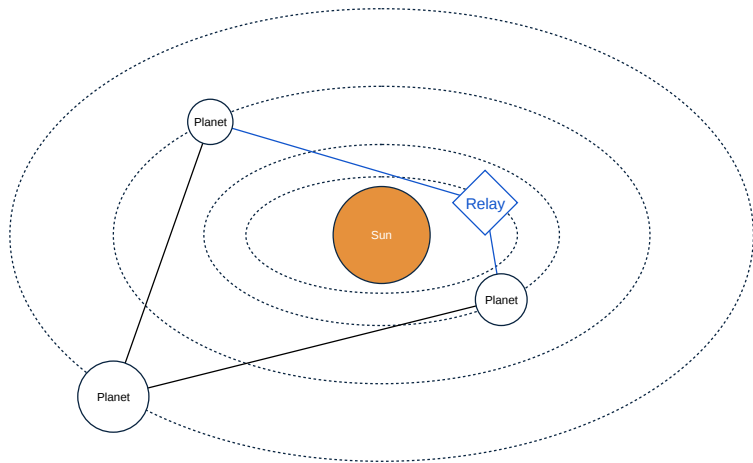
Procedural generation of planetary systems



Connecting systems



Connecting systems



Generating multiple systems

300 planetary systems



CPU usage increase from 2% to 20% in the Linode

Separating world structure from dynamic data

```
{  
  "systems": [  
    {  
      "planets": [  
        {  
          "id": 0,  
          "x": 90,  
          "y": 112,  
          "radius": 20,  
          "owner": 0,  
          "ships": 4,  
          "connections": [1, 2],  
          // ...  
        },  
        // ...  
      ]  
    },  
    // ...  
  ],  
  // ...  
}
```

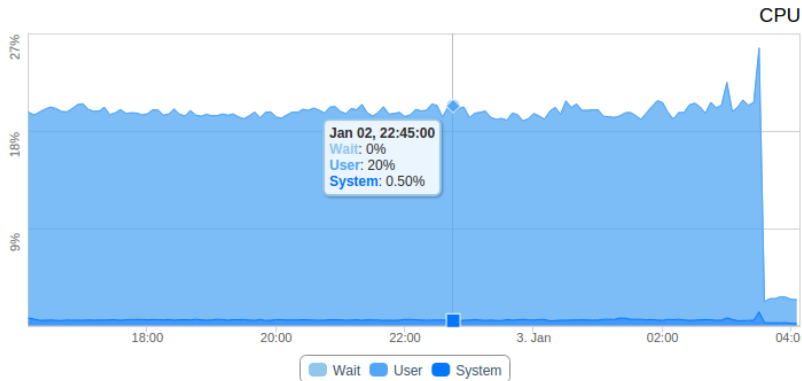
A snapshot before the optimization

Separating world structure from dynamic data

```
[  
  0, 4, // ...  
]
```

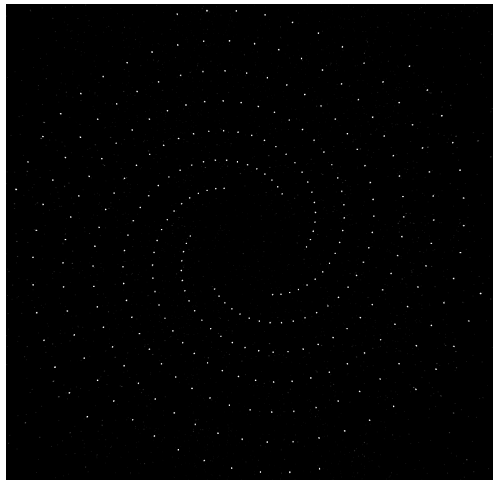
A snapshot after the optimization

Separating world structure from dynamic data



CPU usage drop after separating world structure from dynamic data

Rendering a galaxy



$$x(t) = at \cos(t + \theta)$$

$$y(t) = at \sin(t + \theta)$$

Rendering a galaxy



Identicons

Rendering a galaxy



Live demo

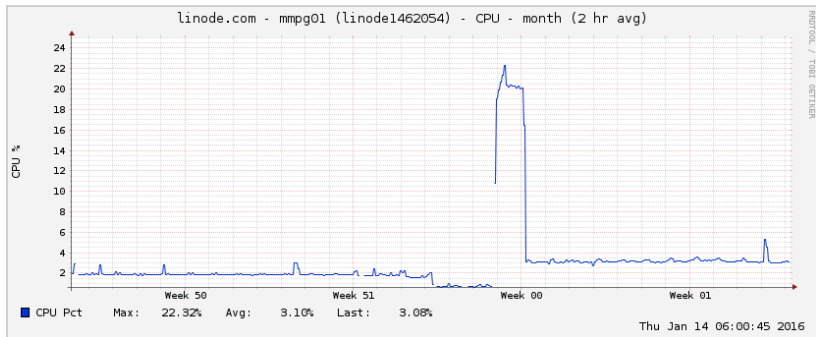
Validation of secondary objectives

- 1 Develop an abstract game engine
- 2 Allow hot-swapping of AIs
- 3 Implement a real-time webviewer
- 4 Make the infrastructure scalable and stable
- 5 Create a game example

Validation of secondary objectives

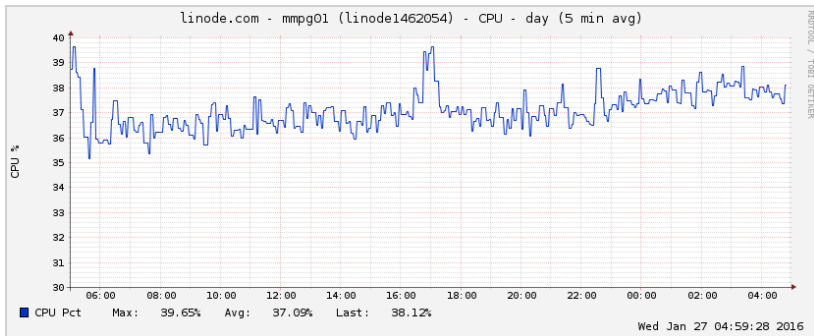
- 1 Develop an abstract game engine ✓
- 2 Allow hot-swapping of AIs ✓
- 3 Implement a real-time webviewer ✓
- 4 Make the infrastructure scalable and stable
- 5 Create a game example ✓

Stability



CPU usage of the platform in a 30 day period (Linode 1024)

Scalability



CPU usage of the platform with 50 players and 300 planetary systems in a day (Linode 1024)

Validation of secondary objectives

- 1 Develop an abstract game engine ✓
- 2 Allow hot-swapping of AIs ✓
- 3 Implement a real-time webviewer ✓
- 4 Make the infrastructure scalable and stable
- 5 Create a game example ✓

Validation of secondary objectives

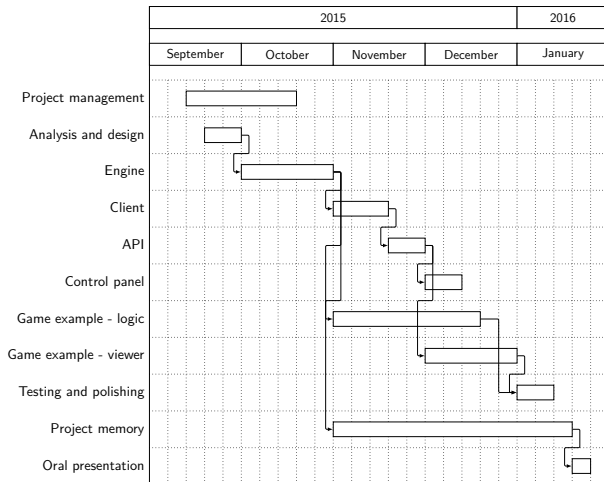
- 1 Develop an abstract game engine ✓
- 2 Allow hot-swapping of AIs ✓
- 3 Implement a real-time webviewer ✓
- 4 Make the infrastructure scalable and stable ✓
- 5 Create a game example ✓

Validation of the main objective

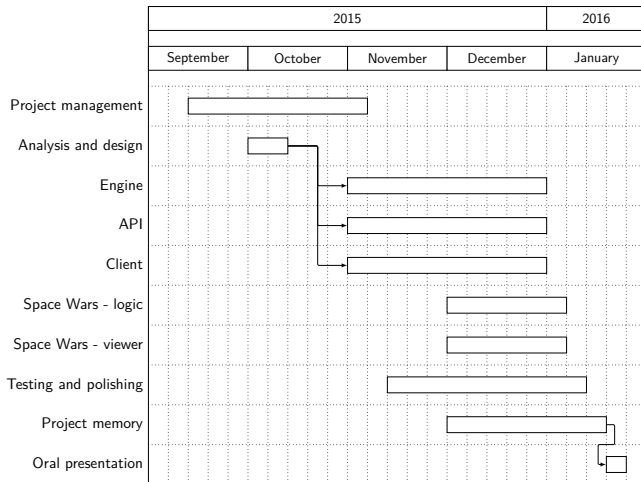
Develop a set of components that ease the creation and the usage of MMPGs.

- Abstract game engine ✓
- Client library ✓
- Documentation ✗

Planning timeline



Final timeline



Time table (h)

Task	Expected	Final
Project management course	75	70
Analysis and design	10	20
Engine	70	100
API	50	30
Client	30	30
Control panel	35	15
Game example	100	100
Testing and polishing	40	40
Project memory	40	50
Oral presentation	10	10
Total	460	465

Total cost (€)

Resource	Budget	Total cost
Hardware	60	90
Software	0	0
Human	14500	14800
Electricity	18	20
Internet	7	7
Contingency (10%)	1500	0.00
Total	$\simeq 16000$	$\simeq 15000$

The future

- Improve documentation
- Implement a widget library
- Create a game template
- Develop a game-logic testing suite

Conclusions

The platform was built, featuring:

- Abstract game engine
- Client library
- AI hot-swapping
- Real-time support
- Match replay
- Game example

The platform sets the foundations for a new type of programming games: the MMPGs, while providing a useful set of components to create them and use them.