Outline

1. Organization of this workshop
2. Computational Intelligence for Games
3. The GVGAI Framework
4. Our GVG Contest
5. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
Outline

1. Organization of this workshop
2. Computational Intelligence for Games
3. The GVGAI Framework
4. Our GVG Contest
5. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
The Workshop (I)

- Professional skills:
  - Reading scientific papers.
  - Practising to present a technical talk.
  - Practising to write scientific texts.
- Practical skills:
  - Improving your programming skills.
  - Learning new algorithms.
  - Solving problems.
- Soft skills:
  - Team work.
  - Taking part in international communities and competitions.
The Workshop (II)

- Groups of 3 students:
  - A name for the group.
  - Full names of all group members.
  - A contact email.
- Each group will create 3 controllers:
  - Heuristic based controller (deadline: 12th November 2014).
  - Reinforcement learning controller. (deadline: 26th November 2014).
  - Nature-inspired controller (deadline: 10th December 2014).
  - Each controller will require:
    - Submission of the controller to the website before the deadline.
    - 3 slides / 5 minutes presentation explaining the submitted controller.
- Each group will deliver:
  - A 15-20 minute presentation on the whole Team Project.
  - A 10-15 pages report.
- Assessment:
  - 1/3: Team Project Presentation.
  - 1/3: Submitted controllers.
The Workshop (III)

6 important dates:

- **22nd October 2014**
  - Today! First contact, introduction to the project, form groups, …
- (**+ 3 weeks**) **12th November 2014**
  - **Deadline** for submitting your first controller (Heuristic controller).
  - You will do a 3 slides / 5 minutes presentation about your controller.
  - We’ll introduce some ideas for the next (RL) controller
- (**+ 2 weeks**) **26th November 2014**
  - **Deadline** for submitting your second controller (RL controller).
  - You will do a 3 slides / 5 minutes presentation about your controller.
  - We’ll introduce some ideas for the next (Nature-inspired) controller.
- (**+ 2 weeks**) **10th December 2014**
  - **Deadline** for submitting your third controller (Nature inspired controller).
  - You will do a 3 slides / 5 minutes presentation about your controller.
- **21st January 2015**
  - **Team Project Presentation**.
- **21st January 2015**
  - **Deadline** for Team Project Report.
Outline

1. Organization of this workshop

2. **Computational Intelligence for Games**

3. The GVGAI Framework

4. Our GVG Contest

5. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
Why games?

‘GAMER’ IS NOT A DIRTY WORD.
Why games?

- Constantly growing industry.

**Video game console hardware/software revenue worldwide from 2012 to 2015 (in billion U.S. dollars)**

Source: Gartner, Statista 2014
Why games?

- Multiple devices.
Intelligence in Games

- AI Gap between industry and academia.
- Industry: behaviour of Non Playable Characters (NPCs).
  - Path planning or pathfinding: how to move intelligently.
Intelligence in Games

- Industry (cont.)
  - Procedural Content Generation (PCG)

- Player believability.

- AI Directors, Computational Narrative, ...
Intelligence in Games

• Academia

• Game theory
  (i.e. Prisoner’s dilemma).

• (Player) Decision Making.
  • Board games: Chess, Checkers, Backgammon, Reversi, Go . . .
  • Video (real-time) games: Racing, Real-Time Strategy, Platformer, Physics-based . . .

• Procedural Content Generation (PCG): generation of games, levels, etc.
• Player experience, player modelling, game mining, BCI for games, etc.
Conferences and journals are great source of information:

- IEEE Conference on Computational Intelligence and Games (CIG).
- AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).
- Foundations of Digital Games (FDG).
- Games tracks in many general AI conferences:
  - EvoGames (EvoStar).
  - Digital Entertainment and Arts (Genetic and Evolutionary Computation Conference - GECCO).
  - Computational Intelligence and Games (Congress on Evolutionary Computation - CEC).

- IEEE Trans. on Computational Intelligence and Artificial Intelligence in Games.
- IEEE Transactions on Evolutionary Computation.
Competitions

- Multiple video-game competitions in the past years.
- (the largest) subset of these is about agent decision making:
General Game Playing (GGP)

- First AIII GGP Competition, by the Stanford University Logic Group.

- > 1s decision time.
General Video-Game Playing (GVGP)

- Actions performed at much higher rate ($\approx 40\text{ms}$).
- Atari 2600 Collection:
  - Arcade Learning Environment (ALE) [1]
  - Evaluation of AI agents in domain-independent environments (55 games).

Pitfall! and Space Invaders, from [Bellemare et al., 2013]

- RL, UCT and MCTS.
- Contingency awareness.
General Video-Game Playing (GVGP)

- Levine et al. (2010) [2] propose creation of new benchmark for GVGP.
- Compliments ALE in two ways:
  - Creation of games in a more general framework.
  - No screen capture analysis needed, information via encapsulated objects.

- Video Game Description Language (VGDL) [3].
  - Benchmark for learning and planning problems.
  - Base for the GVG-AI Framework, used in this workshop and in the GVG-AI Competition.
Outline

1. Organization of this workshop
2. Computational Intelligence for Games
3. The GVGAI Framework
4. Our GVG Contest
5. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
The GVGAI Website

http://www.gvgai.net

The General Video Game AI Competition - 2014

The 2014 GVG-AI Competition is over! The winner is adrienctx (Adrien Couetoux, Academia Sinica, IIS, TW).

The complete and final results are available here!

Also, the validation set games have been made public. You can watch videos about these games (plus their explanations) here.

The validation set games have been added to the GVG-AI framework, which you can download from the software page.

Welcome to the General Video Game AI Competition webpage. The GVG-AI Competition explores the problem of creating controllers for general video game playing. How would you create a single agent that is able to play any game it is given? Could you program an agent that is able to play a wide variety of games, without knowing which games are to be played?

In this website, you will be able to participate in the General Video Game AI Competition, that will be held at CIG 2014. This competition is run by Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul and Simon Lucas. You will be able to download the starter kit for the competition and submit your controller to be in the rankings.
The GVGAI Website

http://www.gvgai.net

Getting Started

Create a Controller for GVGAI
1. Get the java-vgdll framework code and documentation.
2. Create a controller following the instructions.
3. Have a look at our Sample Controllers for inspiration.
4. Check framework documentation and competition rules.

Submit it and Get in the Rankings
1. Sign up in this website to participate, play and submit.
2. Submit (or update) your controller for evaluation.
3. Your controller will be introduced in the rankings.
4. Join our Google group for updates and discussions.

Useful links

Quick Start:
Getting started, Get the Code

The GVG-AI Framework:
Code, VGDLL, Creating Controllers, Forward Model, Specifications

The GVG-AI Competition:
Bot Competition, Evaluation, Starter Kit Games

Participant area:
Log in
Introduction

Java, Unix.

- **examples → gridphysics**: Game and levels files.
- **controllers**: Sample working controllers.
- **core**: Core codebase of the framework.
- **core.competition**: Competition parameters.
- **core.content**: Game and sprite creation.
- **core.game**: Game engine, Forward Model and Observations.
- **core.player**: Abstract class for players.
- **core.termination**: Game termination conditions.
- **ontology**: Definitions for sprites, avatars, physics and events.
- **tools**: Several useful classes.
- **Test.java**: Entry point to run the framework.
VGDL - Game Definitions

BasicGame

.SpriteSet
  base > Immovable color=WHITE img=base
  avatar > FlakAvatar stype=sam
  missile > Missile
    sam > orientation=UP color=BLUE singleton=True img=spaceship
    bomb > orientation=DOWN color=RED speed=0.5 img=bomb
  alien > Bomber stype=bomb prob=0.01 cooldown=3 speed=0.8 img=alien
  portal >
    portalSlow > SpawnPoint stype=alien cooldown=16 total=20 img=portal
    portalFast > SpawnPoint stype=alien cooldown=12 total=20 img=portal

LevelMapping
  0 > base
  1 > portalSlow
  2 > portalFast

TerminationSet
  SpriteCounter stype=avatar limit=0 win=False
  MultiSpriteCounter stype1=portal stype2=alien limit=0 win=True

InteractionSet
  avatar EOS > stepBack
  alien EOS > turnAround
  missile EOS > killSprite
  missile base > killSprite
  base bomb > killSprite
  base sam > killSprite scoreChange=1
  base alien > killSprite
  avatar alien > killSprite scoreChange=-1
  avatar bomb > killSprite scoreChange=-1
  alien sam > killSprite scoreChange=2
VGDL - Level Definitions
A Sample (random) controller

```java
package random; //The package name is the same as the username in the web.

public class Agent extends AbstractPlayer {

    protected Random randomGenerator;

    //Constructor. It must return in 1 second maximum.
    public Agent(StateObservation so, ElapsedCpuTimer elapsedTimer) {
        randomGenerator = new Random();
    }

    //Act function. Called every game step, it must return an action in 40 ms maximum.
    public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {

        //Get the available actions in this game.
        ArrayList<Types.ACTIONS> actions = stateObs.getAvailableActions();

        //Determine an index randomly and get the action to return.
        int index = randomGenerator.nextInt(actions.size());
        Types.ACTIONS action = actions.get(index);

        //Return the action.
        return action;
    }
}
```
StateObservation (I)

- Allows the agent to query the state of the game:
  - StateObservation.getGameScore();
  - StateObservation.getGameTick();
  - StateObservation.getGameWinner();
  - StateObservation.isGameOver();
  - StateObservation.getWorldDimension();
- ...the state of the avatar:
  - StateObservation.getAvatarPosition();
  - StateObservation.getAvatarSpeed();
  - StateObservation.getAvatarOrientation();
  - StateObservation.getAvatarResources();
- ...the available actions in the game:
  - StateObservation.getAvailableActions();
- ...the history of events (collisions) in the game:
  - StateObservation.getEventsHistory();
... *Observations* in the game:
- StateObservation.getObservationGrid();
- StateObservation.getNPCPositions(position?);
- StateObservation.getImmovablePositions(position?);
- StateObservation.getMovablePositions(position?);
- StateObservation.getResourcesPositions(position?);
- StateObservation.getPortalsPositions(position?);
- StateObservation.getFromAvatarSpritesPositions(position?);

What is an *Observation*? It is an object that contains:
- *int itype*: Type of sprite of this observation.
- *int category*: Category of this observation (static, resource, npc, etc.).
- *int obsID*: Unique ID for this observation.
- *Vector2 position*: Position of the observation.
- *double sqDist*: Distance from this observation to the reference.
StateObservation (III)

In *StateObservation*:

1. `ArrayList<Observation>[][] getObservationGrid();`

- Bi-dimensional array, matching the level grid.
- Each ArrayList contains list of Observations in that position.

2. `ArrayList<Observation>[][] getNPCPositions();`

Returns a list of observations of NPC in the game.

3. `ArrayList<Observation>[][] getNPCPositions(Vector2d reference);`

- As there can be NPCs of different type, each entry in the array corresponds to a sprite type.
- Every ArrayList contains a list of objects of type Observation.
- If “reference” is given, Observations are sorted by distance to the reference’s position.
The Forward Model

Also, in *StateObservation*:

1. `StateObservation copy();`

   - Create a copy of the *StateObservation* object.

2. `void advance(Types.ACTIONS action);`

   - Advances the StateObservation object, applying the *action* supplied.
   - Allows to simulate the effects of applying actions.
   - The StateObservation updates itself to reflect the next state.
   - **Important**: The games are stochastic in nature!
     - The next state must be considered as a *possible* future state when applying a certain action.
     - The agent has the responsibility to deal with these *inaccuracies*. 
Outline

1. Organization of this workshop
2. Computational Intelligence for Games
3. The GVGAI Framework
4. Our GVG Contest
5. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
Our GVG Contest - www.gvgai.net - Sets and steps

- There are three different sets where you can submit your controller.
- You must submit to all of them before each upcoming deadline.
  - Two sets are for training, and their games are included in the framework.
  - The third set is for test, and you won’t know the games at all!

1. Sign up (one account per controller).
   - eg: GROUPNAME_HR, GROUPNAME_RL, GROUPNAME_NI
   - For now, just create one account with the username GROUPNAME_HR.


   - **Important:** See the instructions on that page:
     - Your agent must be implemented in a file called `Agent.java`.
     - The Java package where it is contained must be **the same as your username**.
     - Your controller will be compiled and executed automatically.

4. Check the rankings:
   - Training Set 1: http://www.gvgai.net/gvg_rankings_ovgu_t1.php.
Our GVG Contest - www.gvgai.net - Rankings

- Evaluation in 10 games, 5 levels per game, 1 times.
- Three results are considered per game, in this order:
  - Number of victories (↑).
  - Total score (↑).
  - Time spent (↓).
- All entries ranked and awarded with points: 25, 18, 15, 12, 10, 8, 6, 4, 2, 1.
- Final rankings by adding all points across all games.

<table>
<thead>
<tr>
<th>Game</th>
<th>Username</th>
<th>Country</th>
<th>Points</th>
<th>Winner</th>
<th>Score</th>
<th>Timesteps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chase</td>
<td>IdealStandard</td>
<td>Switzerland</td>
<td>25</td>
<td>5</td>
<td>50</td>
<td>349</td>
</tr>
<tr>
<td>Chase</td>
<td>JinJerry</td>
<td>Taiwan, ROC</td>
<td>18</td>
<td>5</td>
<td>49</td>
<td>665</td>
</tr>
<tr>
<td>Chase</td>
<td>adrienctx</td>
<td>Taiwan, ROC</td>
<td>15</td>
<td>5</td>
<td>48</td>
<td>2166</td>
</tr>
<tr>
<td>Chase</td>
<td>FastEvoMCTS</td>
<td>United Kingdom</td>
<td>12</td>
<td>4</td>
<td>48</td>
<td>2367</td>
</tr>
<tr>
<td>Chase</td>
<td>MIMCTS</td>
<td>United Kingdom</td>
<td>10</td>
<td>4</td>
<td>47</td>
<td>2652</td>
</tr>
<tr>
<td>Chase</td>
<td>MIBot</td>
<td>Korea, Republic of</td>
<td>8</td>
<td>4</td>
<td>39</td>
<td>5610</td>
</tr>
<tr>
<td>Chase</td>
<td>sampleMCTS</td>
<td>United Kingdom</td>
<td>6</td>
<td>3</td>
<td>37</td>
<td>6196</td>
</tr>
<tr>
<td>Chase</td>
<td>sampleGA</td>
<td>United Kingdom</td>
<td>4</td>
<td>3</td>
<td>24</td>
<td>8996</td>
</tr>
<tr>
<td>Chase</td>
<td>ssamot</td>
<td>United Kingdom</td>
<td>2</td>
<td>2</td>
<td>37</td>
<td>7305</td>
</tr>
<tr>
<td>Chase</td>
<td>T2Thompson</td>
<td>United Kingdom</td>
<td>1</td>
<td>2</td>
<td>23</td>
<td>5666</td>
</tr>
<tr>
<td>Chase</td>
<td>random</td>
<td>United Kingdom</td>
<td>0</td>
<td>1</td>
<td>22</td>
<td>4910</td>
</tr>
<tr>
<td>Chase</td>
<td>Normal_MCTS</td>
<td>Korea, Republic of</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>8820</td>
</tr>
</tbody>
</table>
Our GVG Contest - 12/11/2014 - Groups

12/11/2014 - HR rankings and groups of the contest:

<table>
<thead>
<tr>
<th>Team</th>
<th>Members</th>
<th>TR1</th>
<th>TR2</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team TopBug</td>
<td>Martin Hünermund and Jens Dieskau.</td>
<td>1st</td>
<td>1st</td>
<td>1st</td>
</tr>
<tr>
<td>Team emergence</td>
<td>Julian Blank and Frederick Sander.</td>
<td>2nd</td>
<td>2nd</td>
<td>2nd</td>
</tr>
<tr>
<td>Team Taccocat</td>
<td>Erik Schondorff, Xenija Neufeld, Florian Uhde.</td>
<td>4th</td>
<td>4th</td>
<td>3rd</td>
</tr>
<tr>
<td>Team xixue</td>
<td>Shengnan Chen, Vishnu Unnikrishnan and Shadi Akhras</td>
<td>3rd</td>
<td>3rd</td>
<td>4th</td>
</tr>
</tbody>
</table>
Outline

1. Organization of this workshop
2. Computational Intelligence for Games
3. The GVGAI Framework
4. Our GVG Contest
5. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
Agent Decision Making

- How to make a decision?

A Markov Decision Process is a tuple \( < S, A, P, R, \gamma > \)
- \( S \): Finite set of all possible states of the game.
- \( A \): Finite set of all possible actions in the game.
- \( P \): A State Transition Probability Matrix \( P_{ss'}^a = \mathbb{P}[s'|s, a] \).
- \( R \): A Reward Function, \( R_{ss'}^a = \mathbb{E}[r|s, a] \).
- \( \gamma \): A discount factor \( \gamma \in [0, 1] \).
One Step Look Ahead

- Try them all, pick the action with the highest reward.
public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
    Types.ACTIONS bestAction = null;
    double maxQ = Double.NEGATIVE_INFINITY;
    SimpleStateHeuristic heuristic = new SimpleStateHeuristic(stateObs);
    for (Types.ACTIONS action : stateObs.getAvailableActions()) {
        StateObservation stCopy = stateObs.copy();
        stCopy.advance(action);
        double Q = heuristic.evaluateState(stCopy);
        if (Q > maxQ) {
            maxQ = Q;
            bestAction = action;
        }
    }
    return bestAction;
}
A Simple Value Function

```java
public double evaluateState(StateObservation a_gameState) {
    boolean gameOver = a_gameState.isGameOver();
    Types.WINNER win = a_gameState.getGameWinner();
    double rawScore = a_gameState.getGameScore();

    if (gameOver && win == Types.WINNER.PLAYER_LOSES)
        return HUGE_NEGATIVE;

    if (gameOver && win == Types.WINNER.PLAYER_WINS)
        return HUGE_POSITIVE;

    return rawScore;
}
```
N-Step Look Ahead

- Build a tree: search exhaustively \( N \) steps in the future.
- Pick the action that leads to the highest reward after these \( N \) steps.
Flat Monte Carlo (Flat MC)

- Iteratively, apply $N$ consecutive actions picked randomly.
- Pick the action that leads to:
  - the highest reward after $N$ steps.
  - the highest **average** reward after $N$ steps, etc.

![Diagram showing state transitions](image)
Upper Confidence Bound for Trees (UCT) (I)

The Multi-Armed Bandit Problem.

- Set of unknown distributions $B = \{R_1, R_2, \ldots, R_k\}$ of rewards.
- Played iteratively, during $H$ action selections.
- Mean values of these reward distributions: $\{\mu_1, \mu_2, \ldots, \mu_k\}$
- The goal is to maximize the sum of rewards (minimizing the loss).
- Each action is pulling one lever. How do you choose?
  - Play randomly.
  - Epsilon-greedy strategies.
  - Many more . . .
Upper Confidence Bound for Trees (UCT) (II)

- Balance exploration and exploitation.
- An example: UCB1.

\[
a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\}
\]

- \(Q(s, a)\): Average of rewards after taking action \(a\) from state \(s\).
- \(N(s)\): Times the state \(s\) has been visited.
- \(N(s, a)\): Times the action \(a\) has been picked from state \(s\).
- \(C\): Balances exploitation (\(Q\)) and exploration (\(\sqrt{\ldots}\)) terms.
  - Application dependant.
  - Typical value, for single player games with rewards normalized in (0, 1): \(\sqrt{2}\).
Flat UCB

- Use UCB1 to select action from the root node (or current state).
- Pick \((N - 1)\) actions at random.
- Repeat iteratively, return action with:
  - the highest reward after \(N\) steps.
  - the highest average reward after 1 step \((Q(s, a))\).
  - the most visited node after 1 step (highest \(a\) for \(N(s, a))\).
  - the highest UCB1 value after 1 step, etc.

![Diagram of Flat UCB](chart)
Building a tree with UCB1

- Build a tree: search $N$ steps in the future.
- The search is **not** exhaustive: tree grows asymmetrically.
- Repeat iteratively, return action with:
  - the highest reward after $N$ steps.
  - the highest average reward after 1 step ($Q(s, a)$).
  - the most visited node after 1 step (highest $a$ for $N(s, a)$).
  - the highest UCB1 value after 1 step, etc.

![Diagram of a tree with UCB1 selection process]
Monte Carlo Tree Search (MCTS)

1. Tree selection (UCB1)
2. Expansion
3. Monte Carlo simulation (Random)
4. Back-propagation

State evaluation

Tree policy

Default policy
Open vs. Closed loop control

- Our methods must face the challenge of stochasticity.
  - If MCTS stores the nodes in the tree: closed loop.
  - This means that, once the node is added, the state is unchanged.
  - Quite possibly, this might lead to inaccuracies ($P_{ss'}^a = \mathbb{P}[s'|s, a]$).
  - It is possible to do statistics on the actions (open loop).

- In Closed loop control:
  - Intermediate states are stored during the algorithm.
  - Every new action is applied over a (previously) stored state.
  - In MCTS, each node stores a state, within the node statistics.

- In Open loop control:
  - Intermediate states are not stored during the algorithm.
  - Every new action is applied over the state reached at that moment.
  - In MCTS, each node stores only the statistics, not the store.
  - In other words, each decision in the tree uses the Forward Model.

- **Summarizing:**
  - Closed loop control works on a sequence of (state, action) pairs.
  - Open loop control works on a sequence of actions.
Evolutionary algorithms (EA)

“Traditional” Evolutionary Algorithms:
- Population of individuals, each one is a solution to a given problem.
- Each individual is evaluated in the problem, and assigned a *fitness*.
- The *fitness* indicates *how good* this solution is to the given problem.
- Population evolves during several generations, creating new individuals by recombination and mutation of individuals in the population.
- Elitism: population does not discard the best individual(s) of a generation.

Possibility, following a “classical” approach for General Video Game Playing:
- Each individual is a player: determines how action decisions are made.
- This is *offline* training:
  - EA determines the best agent first.
  - The *submitted* agent does not use *online* evolution.
Rolling Horizon Evolutionary Algorithms (RH-EA)

No offline training, evolution is use online, while playing the game.

- Each individual is a sequence of actions to apply from the current state: it is open loop control.
- Fitness: Evaluation of the state reached after applying the sequence of actions.
- Within the real-time constraints, the RH-EA evolves the best sequence of actions.
- Evolve the population normally: mutation, crossover, elitism, etc.
- When the process is over, apply first action of the best individual.
- SampleGA Sample Controller.
Some additional ideas

Other things that can be included:

- **Reducing the search space:**
  - Macro-actions.
  - Avoid consecutive opposite actions.
  - Avoid actions that do not change the state.

- **Knowledge discovery:**
  - Use the states visited during the simulations to discover game features.
    - i.e.: what events (collisions) seem to award points?
    - i.e.: what events make the avatar win/lose the game?
  - **Maximize exploration:**
    - i.e.: reward states that visit newly found positions.
    - i.e.: reward states with events rarely seen.
  - Incorporate this into the state evaluation function.

- **New ideas?**
  - Neural Networks.
  - Particle Swarm Optimization.
  - Ant Colony Optimization.
TeamProject Report - Due 21st January 2015

- Two templates provided, see TeamProject website: Word and LaTeX.
- 15-20 pages.
- **Suggested** structure:
  - Introduction.
  - Literature Review.
  - Background.
  - Techniques Implemented.
  - Experimental Study.
  - Conclusions and Future Work.
  - References.

- Results to report:
  - On the three controllers implemented.
    - Comparisons **must** be as fair as possible: state evaluation, look-ahead depth, etc.
    - Make clear this in the report. If something cannot be made equal, explain why.
  - Results in the final rankings on the website (you can keep submitting!).
  - **More importantly:** results comparing your three controllers.
    - Victory rate, score, time steps.
On 21st January 2015 ... TeamProject Presentations

- Each team gives a 15-20 minutes presentation.
- Describe all controllers implemented.
  - Brief description of the core techniques used (MCTS, EA?).
  - Explain how they have been adapted for GVGAI.
- Report results.
- Development process:
  - What things worked? What things didn’t work? Was something surprising (in either way)?
  - Evolution of the State Evaluation function: how has it changed from controller to controller.
- Main problems and difficulties found.
- Conclusions: main findings and lessons learned.
- Future work.

- Ahh... and make it interesting! (= fun).
Marking breakdown

- 40% Final Presentation.
  - Clarity, time, all points above, etc.

- 40% Final Report.
  - Clarity, explanations, references, all points above, etc.

- 20% Code.
  - Quality: re-usability, clarity, comments, efficiency, etc.
Some Readings

MCTS Survey:


About MCTS for GVG and other real-time games. Also macro-actions:


Scientific writing:

References for General Video Game Playing

