AI researchers: Games are your friends!

Julian Togelius
New York University
julian.togelius.com
@togelius
Who am I?

• From Malmö, Sweden

• Studied: Lund (Sweden) >> Sussex (UK) >> Essex (UK)

• Worked: Lugano (Switzerland) >> Copenhagen (Denmark) >> New York (USA)

• philosophy + psychology >> artificial intelligence + robotics >> games

• Current research focus: AI in games (player modeling, procedural content generation, evolutionary computation)
Artificial Intelligence
Artificial Intelligence

Making computers able to do things which currently only humans can do.
What do humans do with games?
What do humans do with games?

• Play them
• Study them
• Build content for them - levels, maps, art, characters, missions…
• Design and develop them
Learning to play board games
AI applied to games

Learning to play board games
Challenges in AI/CI in Video Games

• Learning to play individual games
• Playing in a human-like / believable manner
• General game playing
• Modeling player experience/style/preference
• Generating game content
• Generating games
• AI-assisted game design tools
Video games as AI testbeds / benchmarks
AI can be used for playing specific games
AI playing games
Mario AI Championship

- Ran 2009-2012

- Started with Gameplay track, which got progressively harder through generating harder levels

- Added three more tracks: Gameplay track, Learning track, Level Generation track, and Turing Test track

All methods have limits
REALM: Evolution to the rescue

Slawomir Bojarski and Clare Bates Congdon: REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario. CIG 2010.
Human-like (?) playing
Tell-tale signs of “humanity”

- Pauses before actions, “hesitates”
- Does entirely uncalled for actions
- Tries something and fails
- Does not jump off the platform at the very last pixel
Car racing

• Driving a car fast requires fine motor control (in both senses)

• Optimizing lap times requires planning

• Overtaking requires adversarial planning
The 2011 Simulated Car Racing Championship @ Evo*-2011

Organizers
Daniele Loiacono, Politecnico di Milano
Luigi Cardamone, Politecnico di Milano
Martin Butz, University of Würzburg
Pier Luca Lanzi, Politecnico di Milano
Learning to drive from humans

Can we construct an AI that can play many games?
General intelligence

According to Legg and Hutter: sum of the performance of an agent on all possible problems, weighted by their simplicity

$$\Upsilon(\pi) := \sum_{\mu \in E} 2^{-K(\mu)} V^\pi_\mu.$$
The general video game playing competition

- Competitors submit controllers (AI programs written in Java)
- The game engine lets these controllers play a number of unseen games, and scores them
- The games are written in the Video Game Description Language
The Video Game Description Language

• Developed in order to be able to represent most games from the Atari 2600 era (and many from the C64 era)

• Assumes 2D movement and graphical logic

• Compact and human-readable

• Game engines in Java and Python
BasicGame

SpriteSet
sward > Flicker color=LIGHTGRAY limit=1 singleton=True img=sword.png
dirt > Immovable color=BROWN img=dirt.png
exitdoor > Door color=GREEN img=door.png
diamond > Resource color= YELLOW limit=10 shrinkfactor=0.75 img=diamond.png
boulder > Missile orientation=DOWN color=GRAY speed=0.2 img=boulder.png
moving >
  avatar > ShootAvatar stype=sword img=avatar.png
  enemy > RandomNPC
  crab > color=RED img=camel.png
  butterfly > color=PINK img=butterfly.png

LevelMapping
.
E > exitdoor
O > boulder
X > diamond
c > crab
b > butterfly

InteractionSet
dirt avatar > killSprite
dirt sword > killSprite
diamond avatar > collectResource
diamond avatar > killSprite scoreChange=2
moving wall > stepBack
moving boulder > stepBack
avatar boulder > killIfFromAbove scoreChange=-1
avatar butterfly > killSprite scoreChange=-1
avatar crab > killSprite scoreChange=-1
boulder dirt > stepBack
boulder wall > stepBack
boulder diamond > stepBack
boulder boulder > stepBack
enemy dirt > stepBack
enemy diamond > stepBack
butterfly crab > transformTo stype=diamond scoreChange=1
exitdoor avatar > killIfOtherHasMore resource=diamond limit=9

TerminationSet
SpriteCounter stype=avatar limit=0 win=False
SpriteCounter stype=exitdoor limit=0 win=True
Human player in Boulder Dash
Random controller on Boulder Dash
Monte Carlo Tree Search
MCTS controller on Boulder Dash
Random controller on “Aliens” (Space Invaders)
MCTS controller on “Aliens” (Space Invaders)
signed as single player games, for ease of initial setup and Two player games:

<table>
<thead>
<tr>
<th>Rank</th>
<th>Username</th>
<th>G-1</th>
<th>G-2</th>
<th>G-3</th>
<th>G-4</th>
<th>G-5</th>
<th>G-6</th>
<th>G-7</th>
<th>G-8</th>
<th>G-9</th>
<th>G-10</th>
<th>Total Points</th>
<th>Victories</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>adrienctx</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>10</td>
<td>15</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>158</td>
<td>256/500</td>
</tr>
<tr>
<td>2</td>
<td>JinJerry</td>
<td>18</td>
<td>6</td>
<td>18</td>
<td>25</td>
<td>15</td>
<td>6</td>
<td>18</td>
<td>18</td>
<td>12</td>
<td>12</td>
<td>148</td>
<td>216/500</td>
</tr>
<tr>
<td>3</td>
<td>SampleMCTS†</td>
<td>10</td>
<td>18</td>
<td>6</td>
<td>4</td>
<td>18</td>
<td>25</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>99</td>
<td>158/500</td>
</tr>
<tr>
<td>4</td>
<td>Shmokin</td>
<td>6</td>
<td>25</td>
<td>0</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>0</td>
<td>77</td>
<td>127/500</td>
</tr>
<tr>
<td>5</td>
<td>Normal_MCTS</td>
<td>12</td>
<td>0</td>
<td>4</td>
<td>15</td>
<td>4</td>
<td>15</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>68</td>
<td>102/500</td>
</tr>
<tr>
<td>6</td>
<td>culim</td>
<td>2</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>61</td>
<td>124/500</td>
</tr>
<tr>
<td>7</td>
<td>MMbot</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>12</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>59</td>
<td>130/500</td>
</tr>
<tr>
<td>8</td>
<td>TESTGAG</td>
<td>0</td>
<td>8</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>25</td>
<td>52</td>
<td>68/500</td>
</tr>
<tr>
<td>9</td>
<td>Yraid</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>15</td>
<td>6</td>
<td>49</td>
<td>93/500</td>
</tr>
<tr>
<td>10</td>
<td>T2Thompson</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>18</td>
<td>18</td>
<td>47</td>
<td>87/500</td>
</tr>
<tr>
<td>11</td>
<td>MnMCTS</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>18</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>47</td>
<td>109/500</td>
</tr>
<tr>
<td>12</td>
<td>SampleGA†</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>40</td>
<td>76/500</td>
</tr>
<tr>
<td>13</td>
<td>IdealStandard</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>39</td>
<td>134/500</td>
</tr>
<tr>
<td>14</td>
<td>Random†</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>78/500</td>
</tr>
<tr>
<td>15</td>
<td>Tichau</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>55/500</td>
</tr>
<tr>
<td>16</td>
<td>OneStepLookAhead†</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>51/500</td>
</tr>
<tr>
<td>17</td>
<td>levis501</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>50/500</td>
</tr>
<tr>
<td>18</td>
<td>LCU_14</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>54/500</td>
</tr>
</tbody>
</table>

**TABLE IV**

Final results of the GVGAI Competition. †denotes a sample controller.
Modern game development
Procedural content generation in games
Rogue
Diablo III
Spelunky
Civilization IV
Why PCG?

• Save development time and effort (money)
• Unleash non-human creativity
• Create endless games
• Create player-adaptive games
• Study game design by formalizing it
What are the challenges?

• *Speed*
  Real-time? Or design-time?

• *Reliability*
  Catastrophic failures break gameplay

• *Controllability*
  Allow specification of constraints and goals

• *Diversity*
  Content looks like variations on a theme

• *Creativity*
  Content looks “computer-generated”
Search-based PCG

• Use evolution (or similar algorithms) to search for good content

• Main issues:
  • How to represent the content so that the content space can be searched effectively
  • How to evaluate the quality of content

J. Togelius, G. Yannakakis, K. O. Stanley and C. Browne
Search-based Procedural Content Generation: a Taxonomy and Survey
IEEE TCIAIG 2011
Let’s evolve levels for Super Mario Bros!
Representation

- A number of “vertical slices” are identified from the original SMB levels
- Levels are represented as strings, where each character corresponds to a pattern

4.2 Putting pieces together

In order to explain the vertical slices and how we combine them into patterns we will use an example with an instance of the Enemy: 3-Horde-pattern [11] (as in figure 2). The instance of the pattern could then be described as a sequence...
Evaluation

- 25 patterns are identified in the original SMB levels
- e.g. enemy hordes, pipe valleys, 3-paths...
- The evaluation function counts the number of patterns found in the level
The one-point crossover is illustrated in figure 7.
How would we generate rules for completely new games?
An example: *Ludi*
creating board games

- Construct a language that can describe games...

- …and a game engine that can play any game described in the language

- Then, use *evolution* to design games!
The Ludi Game
Description Language

• In practice limited to board games

• *Ludeme*: Fundamental units of independently transferable game information (“game meme”)
  
  • (tiling square)
  
  • (size 3 3)
Tic-Tac-Toe

(game Tic-Tac-Toe
 (players White Black)
 (board
  (tiling square i-nbors)
  (size 3 3)
 )
 )
 (end (All win (in-a-row 3)))
)
The term game shall henceforth refer to a two-player combinatorial game throughout this paper. Such games are an ideal test bed for the experiments as they are typically deep but described by simple, well-defined rule sets. Note that this is not a work in combinatorial game theory (CGT), which is concerned with the analysis of games with a view to solving them or at least finding optimal strategies [3] and developing artificial players able to challenge human experts. Within the context of this study, the artificial player is of little interest except as a means for providing self-play simulations. While it must be of sufficient strength to provide meaningful playouts, we are concerned primarily with the quality of the game itself rather than the quality of the player.

B. Ludemes

Just as a meme is a unit of information that replicates from one person to another [4], a ludeme is a game meme or unit of game information. First coined by Borvo [5], this term describes a fundamental unit of play often equivalent to a rule; ludemes are the conceptual equivalent of a game's components – both material and non-material – and are notable for their ability to pass from one game or game class to another [6].

Ludemes may be single units of information, such as the following items that describe aspects of the game board shown in Fig. 1(a):

- (tiling square) (size 3 3)

Conceptually related items may be encapsulated to form higher level compound ludemes as follows:

- (board (tiling square) (size 3 3))

Collecting rules into such compound ludemes is a convenient way to describe games. For example, the essence of Tic-Tac-Toe may be succinctly described as follows (assuming a two-player combinatorial model):

- (game Tic-Tac-Toe (board (tiling square) (size 3 3)) (win (in-a-row 3)))

The concept of an entire game as an item of information may seem odd but it is valid; there exist many examples of identical games being discovered, fully formed, at similar times. The most famous case is the independent discovery of Hex by mathematicians Piet Hein and John Nash in the 1940s [201]. A more recent example is Chameleon, discovered by New Zealand and USA designers within a week of each other in 2003. Such cases may be examples of "memetic convergence" in action towards optimal designs.

C. Recombination Games

Given a game in its ludemic form, it is a simple matter to manipulate its rules to create variants and new games. For Tic-Tac-Toe, such modifications might include the board size:

- (size 2 2)

or the target line length:

- (win (in-a-row 2))

However, a moment's reflection will reveal that each of these changes break the game, by making it unwinnable in the first case and trivially winnable in the second. Other manipulations might involve extending the board to three dimensions, as shown in Fig. 1(b):

- (size 3 3 3)

or inverting the end condition to give a misere version:

- (lose (in-a-row 3))

These variants are both more interesting but still trivially solvable, and are more notable for their novelty value than any inherent value as games. There is much room for improvement in this branch of the N-in-a-row family. The difficulty of deriving an interesting game from Tic-Tac-Toe does not just stem from the fact that it is itself flawed (it is drawish if played correctly). There is the serious problem that rule sets for combinatorial games tend to be highly optimised and fragile; authors strive for the simplest rule sets that give the deepest playing experience, and the slightest change will generally break a game. As in most creative fields, it is easy to generate artificial content but much more difficult to generate artificial content of human expert quality.
Automatic Game Design

- Simple Pac-Man like games
- Rule encoding: what happens when things collide
- Fitness function: learnability

Discovering interesting game variants

Aaron Isaksen, Dan Gopstein, Julian Togelius and Andy Nealen: Discovering Interesting Game Variants. ICCC 2015.
Varying two dimensions

- Lanky Bird
- Chunky Bird
- Tiny Bird
- Squishy Bird
Evolving far-apart games
Evolving far-apart games
Needle Gnat

Lazy Blimp

Droppy Brick

Pogo Pigeon

Figure 7: The four game variants discovered using the Most Unique evolution method with $k = 4$. This method searches for the $k$ games which maximizes the minimum distance between any two points in the set. The games are generated by the authors, not by the algorithm. (a) Needle Gnat: tiny player trying to thread a tight horizontal space. (b) Lazy Blimp: slow moving blimp-like player with minimal gravity and jump. (c) Droppy Brick: frequent rise and fall with high gravity. (d) Pogo Pigeon: very tall, thin bird that frequently hops to avoid crashing into the ground.
Collaborating with the AI

• The AI can design levels (and games)

• But so can you!

• Maybe you have different strengths and can work together?
Generate Level Samples
Adaptive games

• Can we use PCG to create games that adapt to the player?

• Adapt to what? Skill, preferences, strategy, playing style…
Player level preferences in Super Mario Bros

- Neuroevolutionary preference learning
- Player experience model 73-92%

C. Pedersen, J. Togelius, G. N. Yannakakis., *Modeling Player Experience for Content Creation* *IEEE TCIAG*, 2010
What can AI do for games?

• Generate complete games, which requires…
• generating game content, which requires…
• evaluating content and game quality, which requires…
• modeling player preference and style, and…
• learning to play arbitrary games
What can games do for AI?

• Provide superb testbeds, that are varied and human-relevant

• Show us how we think

• Teach us how to create AI that has fun
Further reading


• Georgios N. Yannakakis and Julian Togelius (2014): A Panorama of Artificial and Computational Intelligence in Games. IEEE Transactions on Computational Intelligence and AI in Games.