Video Game Description Language (VGDL) and the Challenge of Creating Agents for General Video Game Playing (GVGP)

Diego Perez, University of Essex (Colchester, UK)
dperez@essex.ac.uk

Tainan, Taiwan, 2015

IEEE Conference on Computational Intelligence in Games

CIG 2015
Outline

1. General (Video) Game Playing

2. The GVGAI Framework

3. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
Outline

1 General (Video) Game Playing

2 The GVGAI Framework

3 Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
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 ($\approx 40$ms).

- Atari 2600 Collection:
  - Arcade Learning Environment (ALE)
  - 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) 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)
  - Benchmark for learning and planning problems.
  - Base for the GVG-AI Framework, used in this workshop and in the GVG-AI Competition.
Outline

1. General (Video) Game Playing

2. The GVGAI Framework

3. Agent Decision Making
   - MCTS and Sampling methods
   - Evolutionary algorithms
The GVGAI Website

http://www.gvgai.net

The General Video Game AI Competition - 2015

CEEC Competition deadline: September 15th, 2015 (23:59, GMT)
CIG Competition results will be announced at CIG-2015, September 2nd, 2015

2015 Roadmap

Follow gvgai

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 GECCO, CIG and CEEC 2015. 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-vgdl 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
- VGD
- Creating Controllers
- Forward Model
- Specifications

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

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
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();
  - StateObservation.getAvatarType();
- ...the available actions in the game:
  - StateObservation.getAvailableActions();
- ...the history of events (collisions) in the game:
  - StateObservation.getEventsHistory();
StateObservation (II)

... 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:
- \textit{int itype}: Type of sprite of this observation.
- \textit{int category}: Category of this observation (static, resource, npc, etc.).
- \textit{int obsID}: Unique ID for this observation.
- \textit{Vector2 position}: Position of the observation.
- \textit{Vector2 reference}: Reference (pivot) position.
- \textit{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.
- 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.

1. `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. General (Video) Game Playing

2. The GVGAI Framework

3. 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 $P_{ss'} = \mathbb{P}[s'|s, a]$.
- $R$: A Reward Function, $R^a_s = \mathbb{E}[r|s, a]$.
- $\gamma$: A discount factor $\gamma \in [0, 1]$. 

![Diagram of Markov Decision Process]
One Step Look Ahead

- Try them all, pick the action with the highest reward.
One Step Look Ahead

```java
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.
Upper Confidence Bound for Trees (UCT)

- 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.
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.
Monte Carlo Tree Search (MCTS)

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

State evaluation
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.
Some Readings

The 2014 GVGAI Competition:


About MCTS for GVG:


About Rolling Horizon Evolutionary Algorithms for GVG:


More papers about GVGAI can be found at our website:

http://www.gvgai.net/papers.php
Example cases

Frogs:
1. SampleMCTS in Frogs with WinScoreHeuristic
2. SampleMCTS in Frogs with SimpleStateHeuristic
3. OneStepLookAhead in Frogs with WinScoreHeuristic
4. OneStepLookAhead in Frogs with SimpleStateHeuristic

Similar with Camel Race.

Interesting challenges:
1. Easy: playing “not” to win at Surround.
2. (Not so) easy: Entry for the GVGAI Competition (CEEC Puzzle games).
The future of the GVGAI Competition

- Tracks:
  - Planning.
  - Learning.
  - Procedural Content Generation (games and levels).

- Enhancing the game space:
  - Two/N Player Games.
  - Continuous Physics.
  - Continuous Action Spaces.
  - Games with Partial Observability.