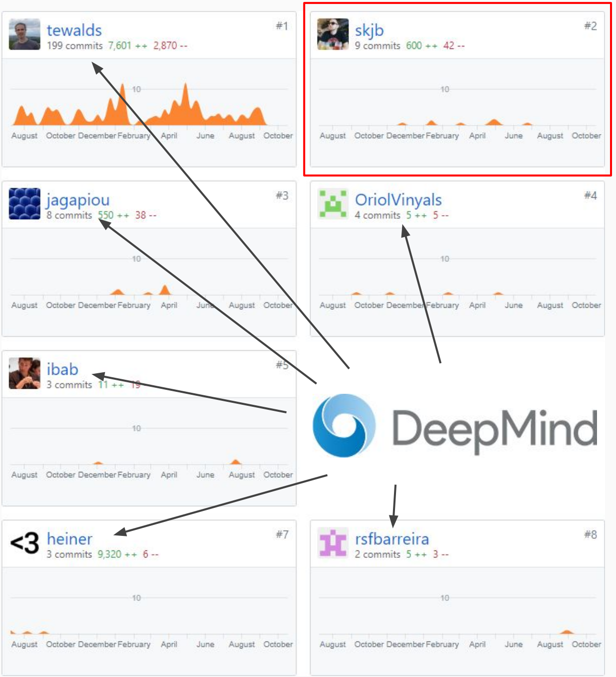
Introduction to PySC2

Steven Brown - Who am I?

- 2nd Highest Code Contributor to PySC2
- Creator of Feature Units



tewalds #1
199 commits 7,601 ++ 2,870 --

skjb #2
9 commits 600 ++ 42 --

jagapiou #3
8 commits 550 ++ 38 --

OriolVinyals #4
4 commits 5 ++ 5 --

ibab #5
3 commits 11 ++ 19

heiner #7
3 commits 9,320 ++ 6 --

rsfbarreira #8
2 commits 5 ++ 3 --

Steven Brown - Where am I?

medium.com/@skjb

twitch.tv/skjb

patreon.com/skjb

What is PySC2?

- StarCraft II machine learning environment
- Interacts with Blizzard’s StarCraft II API
- Python based
- Designed to emulate human abilities
- Backed by DeepMind
- github.com/deepmind/pysc2

What is Blizzard’s StarCraft II API?

- Allows you to interact with the game via Protobuf
- Has limited Linux support
- Can play replays
- Provides the ability to investigate game state
- Provides the ability to perform player actions
- Supports 2 players
- Has some limitations
- Is still being developed

Alternatives to PySC2

- Dave Churchill’s CommandCenter - C++ framework for BW and SC2, very popular and used for bot battles
- Python SC2 - less human realistic
- C#, Clojure, Java, Go

Why PySC2?

- Python = TensorFlow and Scikit
- DeepMind development team
- You want to build a human-comparable bot
- You can’t stand the thought of interacting with other humans or having any spare time, and you want nothing more than to be coding bots all day

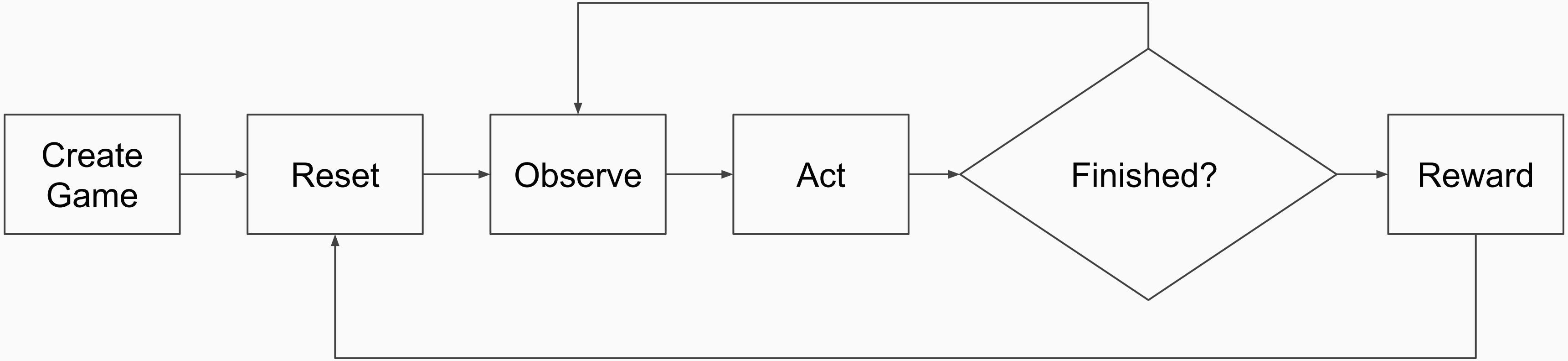
Why StarCraft II?

- Deterministic - there's no random chance, in the same state the same action will produce the same result
- Extremely Large State and Action Space - it takes a lot of time to explore and find what works and what doesn't
- Hidden Information - there is information about your opponent you don't know, exacerbated by fog of war

What sort of agent do you want to build?

- Completely scripted
- Partially scripted, partially ML
- Completely ML but structured specifically for SC2
- Completely ML with no SC2 specific structures

Game Engine Flow

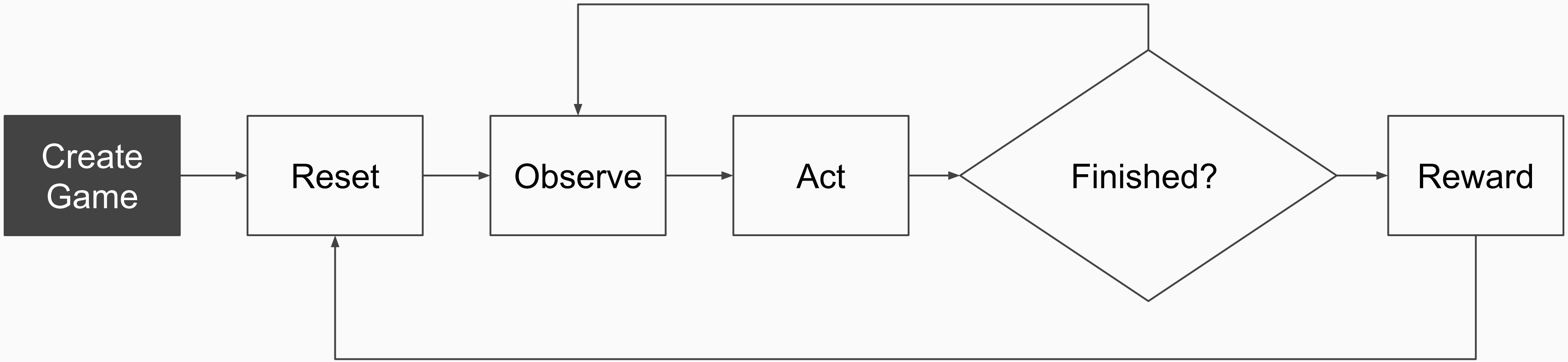


Remind you of anything?

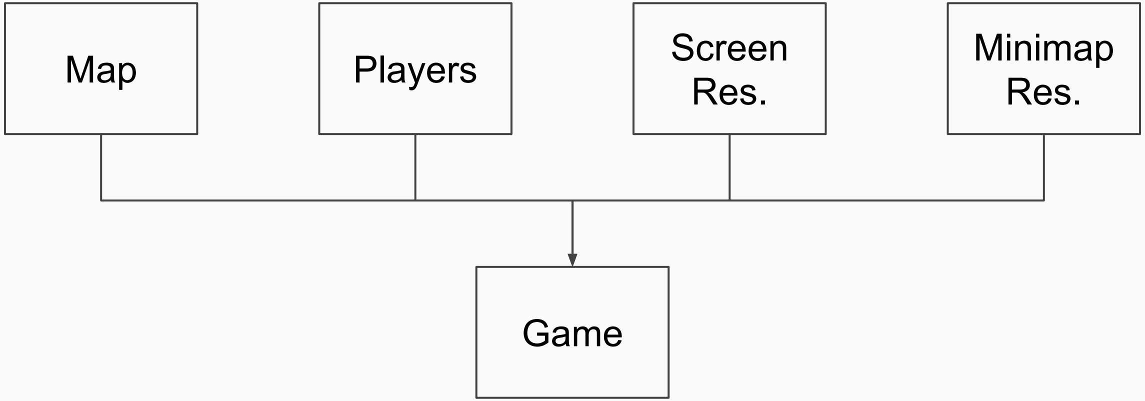
Example Agent

```
class MyAgent(base_agent.BaseAgent):
  def __init__(self):
    super(MyAgent, self).__init__()
    # One-time setup

  def reset(self):
    super(MyAgent, self).reset()
    # Before each game

  def step(self, obs):
    super(MyAgent, self).step(obs)

    # Read state from obs and ALWAYS act
    return actions.FUNCTIONS.no_op()
```

Game Engine Flow



Main Game Options



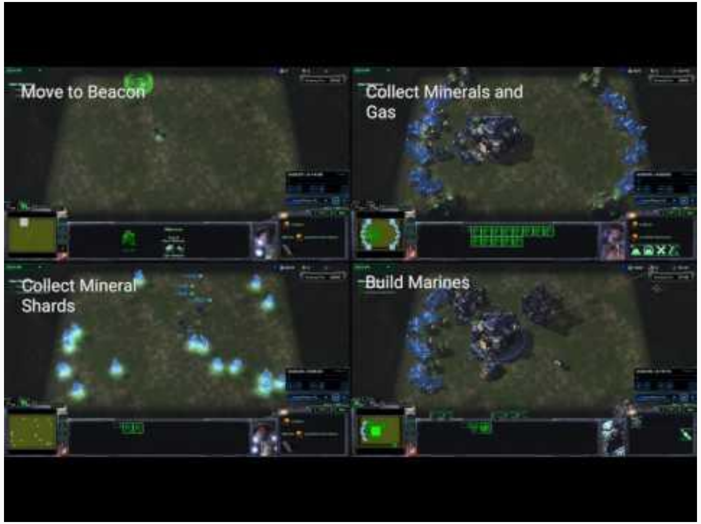
Code Example of Game Options

```
with sc2_env.SC2Env(
    map_name="Simple64",
    players=[sc2_env.Agent(sc2_env.Race.terran),
             sc2_env.Bot(sc2_env.Race.zerg, sc2_env.Difficulty.easy)],
    agent_interface_format=features.AgentInterfaceFormat(
        feature_dimensions=features.Dimensions(screen=84,
                                               minimap=64),
        action_space=actions.ActionSpace.FEATURES),
    ) as env:
```

Maps - Mini Games

Simplified games designed for testing algorithms

- BuildMarines
- CollectMineralsAndGas
- CollectMineralShards
- DefeatRoaches
- DefeatZerglingsAndBanelings
- FindAndDefeatZerglings
- MoveToBeacon



Maps - Melee Maps

- Designed for full game play
- Empty128
- Flat32/48/64/96/128 - No terrain variations
- Simple64/96/128 - Some terrain

Maps - Ladder Maps

- The usual ladder maps
- Can lag behind a bit, depending on Linux support and code updates
- Far more complex to generalise for due to variations in terrain, base locations, resources, etc.

Bot Difficulties

- very_easy
- easy
- medium
- medium_hard = Hard
- hard = Harder
- harder = Very hard
- very_hard = Elite
- cheat_vision
- cheat_money
- cheat_insane

Other Useful Game Options

step_mul.

The number of steps to take before acting again, there are 22.4 steps per second for normal “faster” games, so a value of 8 = 168 APM, I have had issues below a step mul of 2

game_steps_per_episode.

The maximum steps to take before the game automatically ends

Other Useful Game Options

`visualize`

Shows a custom rendered version of the observations, good for debugging but slows the game considerably

`disable_fog`

Disables the fog of war so everything is visible

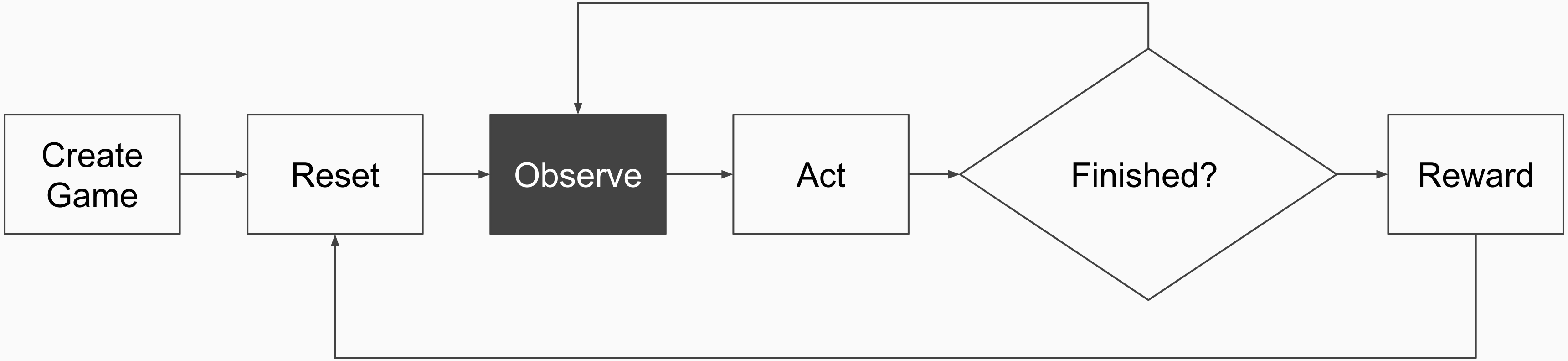
Machine Learning Tips

- Start against very easy bots
- Play as a single race
- Play against a single race
- Disable the fog of war
- Use a single, simple map
- Human players can beat very easy with an APM <60 so consider a step_mul of 20-30

Machine Learning Tips

- Limit the games to half an hour or so (40,320 steps)
- Consider having your agent play against a simple scripted bot
- Consider self-play, you get twice the learning per game but it may not generalise
- Compare your bot against one that chooses completely random actions

Game Engine Flow



Example State

```
state = (command_center_count,
         supply_depot_count,
         barracks_count,
         scv_count,
         marine_count,
         base1_enemy_count,
         base2_enemy_count,
         base3_enemy_count,
         base4_enemy_count,
         base1_friendly_count,
         base2_friendly_count,
         base3_friendly_count,
         base4_friendly_count)
```

Step “obs”

`obs.first()`

Whether or not this is the first step of the game, good for doing things like position and race detection

`obs.last()`

Whether or not this is the last step, use this to learn, save, and wrap things up

`obs.observation.reward`

Use this for sparse rewards, will be 0 for mid-steps or a draw, -1 for a loss, 1 for a win

Step “obs”

`obs.observation.game_loop`

The current step, you can work out the current second if you divide by 22.4

`obs.observation.available_actions`

Contains a list of actions that can be performed in the current state

`obs.observation`

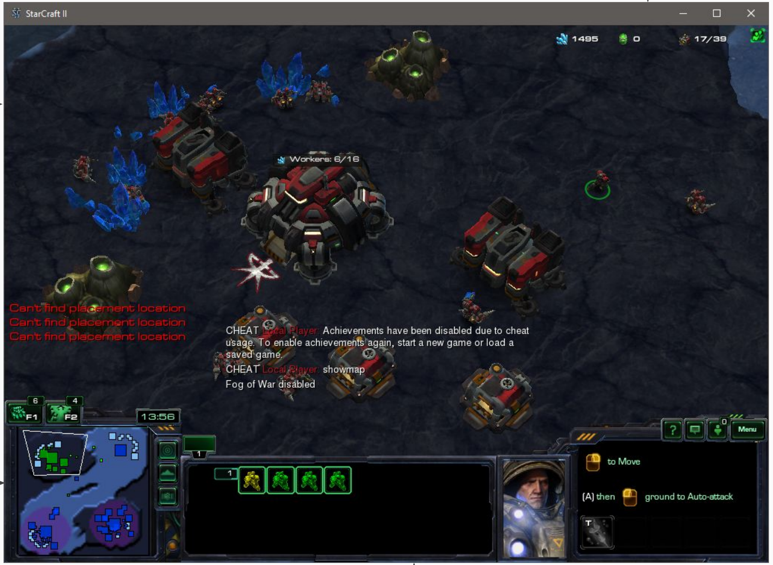
Everything else

Main “Features”

Player

Screen

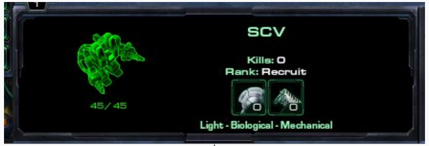
Minimap



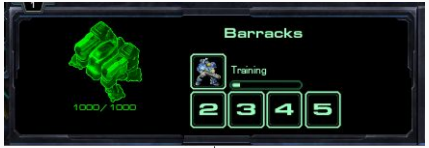
Can't find placement location
Can't find placement location
Can't find placement location

CHEAT Local Player: Achievements have been disabled due to cheat usage. To enable achievements again, start a new game or load a saved game.
CHEAT Local Player: showmap
Fog of War disabled

Multi-select



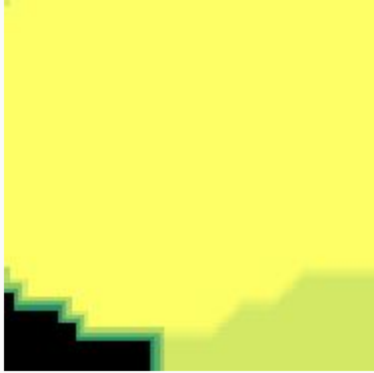
Single-select



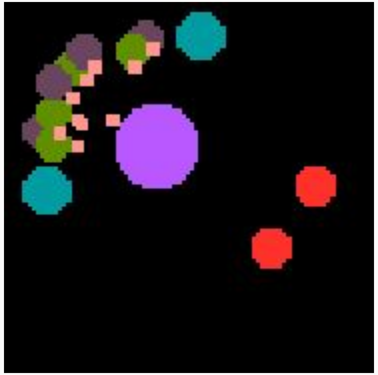
Build Queue

*Cargo not shown

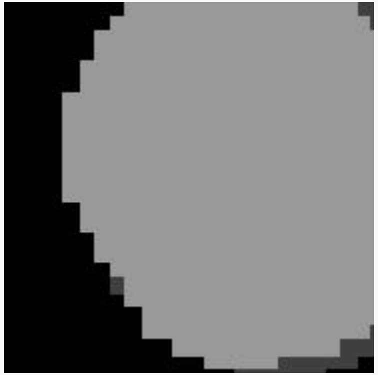
Some Screen Features



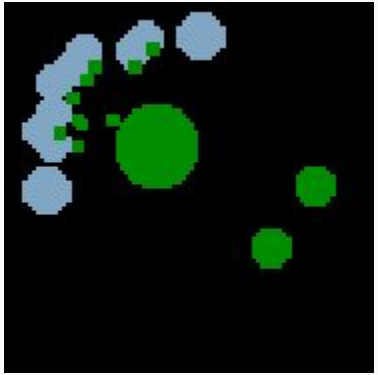
Height map



Unit type



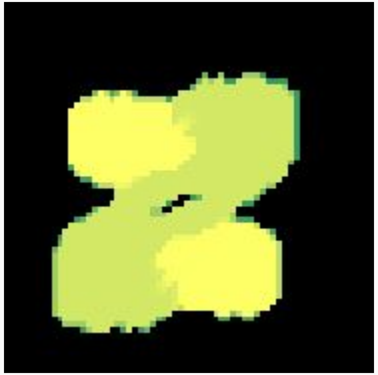
Visibility



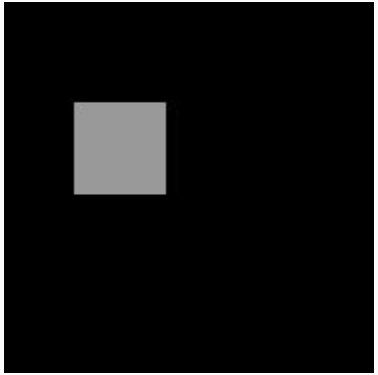
Player relative

Other important features include power, creep and effects

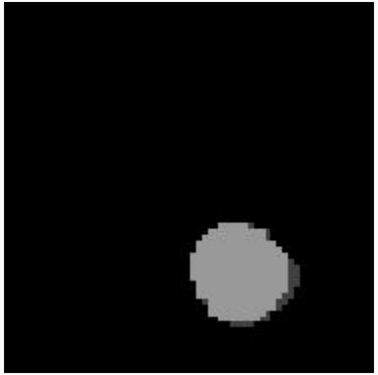
Some Minimap Features



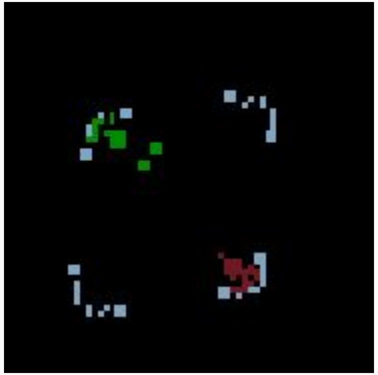
Height map



Camera location



Visibility



Player relative

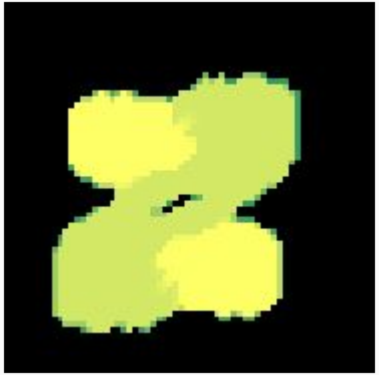
Other important features include creep

Scalar Feature Layers

x ↓

y →

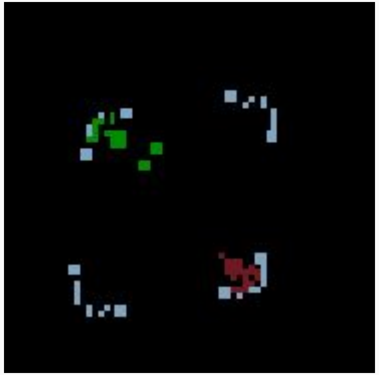
```
[[  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0 255 255 255 212 212   0   0]
 [  0 255 255 255 212 212   0   0]
 [  0   0 212   0 212   0   0   0]
 [  0 212 212 255 255 255   0   0]
 [  0 212 212 212 255 255   0   0]
 [  0   0   0   0   0   0   0   0]]
```



```
height_map = obs.observation.feature_minimap.height_map
height_map[y][x]
height_map[2][1] = 255
```

Categorical Feature Layers

```
[[0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 3 1 0 3 3 0 0]
 [0 3 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 3 0 0 0 3 0 0]
 [0 3 3 0 3 3 0 0]
 [0 0 0 0 0 0 0 0]]
```

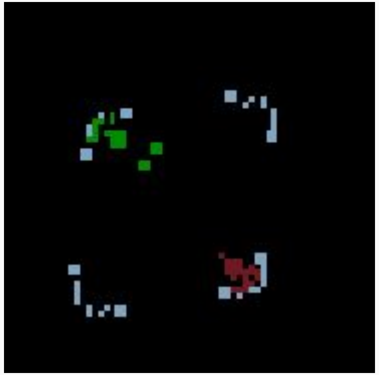


```
player_relative = obs.observation.feature_minimap.player_relative
```

Remember not to use ranges (e.g. 0-3) for categorical layers when feeding into neural networks, instead supply each category as 0 or 1.

Categorical Feature Layers

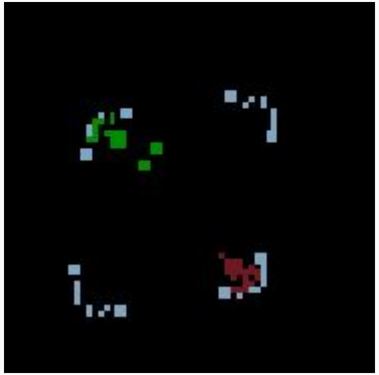
```
[[False False False False False False False False]
 [False False False False False False False False]
 [False False  True False False False False False]
 [False False  True False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]
 [False False False False False False False False]]
```



```
player_relative = obs.observation.feature_minimap.player_relative
minimap_self = (player_relative == features.PlayerRelative.SELF)
```

Categorical Feature Layers

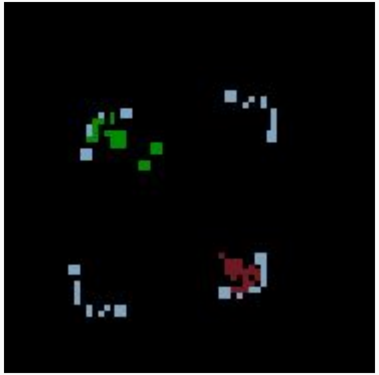
```
([2, 2], [2, 3]) # [y1, y2], [x1, x2]
```



```
player_relative = obs.observation.feature_minimap.player_relative
player_y, player_x = (player_relative == features.PlayerRelative.SELF).nonzero()
```

Categorical Feature Layers

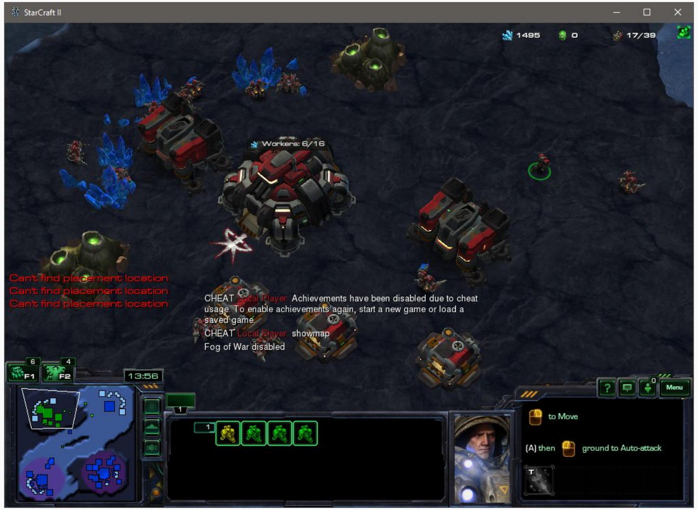
```
([2, 2], # [x1, y1]
 [3, 2]) # [x2, y2]
```



```
player_relative = obs.observation.feature_minimap.player_relative
player_y, player_x = (player_relative == features.PlayerRelative.SELF).nonzero()
player_xy = zip(player_x, player_y)
```

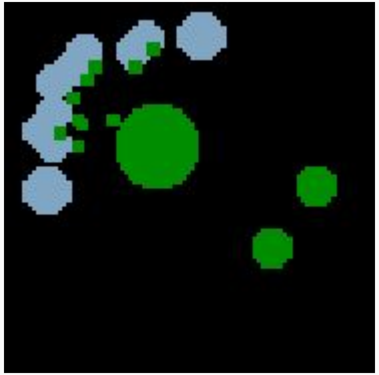
Screen Feature Notes

- The perspective is different, so don't expect everything to perfectly match the normal game
- Units can overlap and be difficult to identify



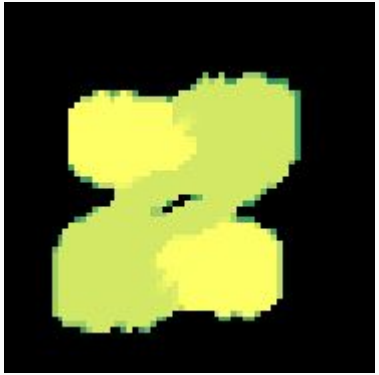
Can't find placement location
Can't find placement location
Can't find placement location

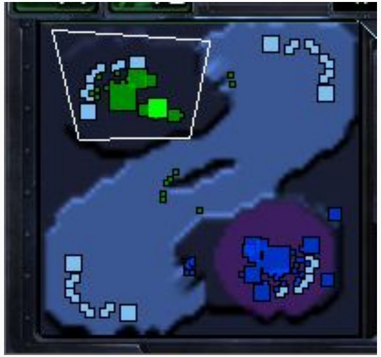
CHEAT Local Player: Achievements have been disabled due to cheat usage. To enable achievements again, start a new game or load a saved game.
CHEAT Local Player: showmap
Fog of War disabled



Minimap Feature Notes

- The minimap in the API may not match the minimap in the game
- Each pixel on the minimap can only contain one detail per feature layer





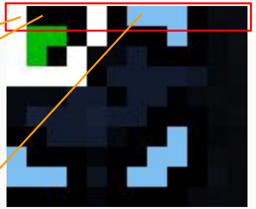
Enable RGB Observations

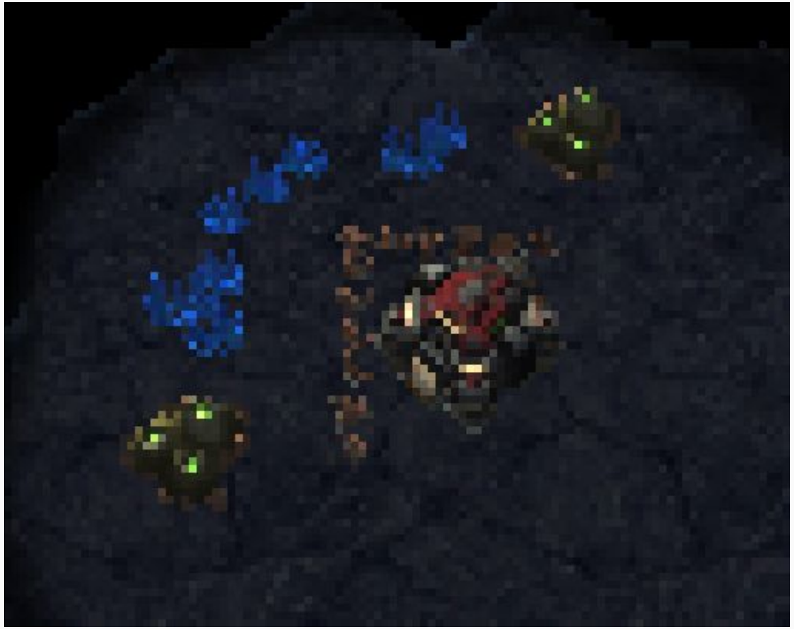
```
with sc2_env.SC2Env(
    map_name="Simple64",
    players=[sc2_env.Agent(sc2_env.Race.terran),
             sc2_env.Bot(sc2_env.Race.zerg, sc2_env.Difficulty.easy)],
    agent_interface_format=features.AgentInterfaceFormat(
        rgb_dimensions=features.Dimensions(screen=84,
                                           minimap=64),
        action_space=actions.ActionSpace.RGB),
    ) as env:
```

RGB Observations

```
[[[255 255 255]
  [  0   0   0]
  [  0   0   0]
  [  0   0   0]
  [255 255 255]
  [  0   0   0]
  [241 191 126]
  [241 191 126]
  [241 191 126]
  [  0   0   0]
  [ 15  12  11]
  [ 17  11  10]]
```

B G R





```
obs.observation.rgb_minimap
obs.observation.rgb_screen
```

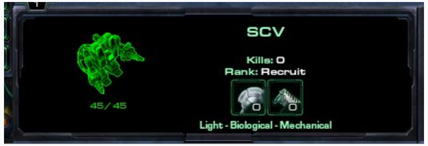
*BGR instead of RGB may be a bug

RGB Feature Notes

- If you are observing in the RGB space you should act in the RGB space to maintain perspective
- Sizing does not seem to be exact (e.g. specifying 8x8 produced a 12x10 grid)
- Screen perspective matches the regular game
- Minimap seems to match the regular game

Single, Multi-Select and Cargo Observations

```
obs.observation.single_select
obs.observation.multi_select
obs.observation.cargo
```



```
[[45
  1
  45
  0
  0
  0
  0]]
```

Unit type (SCV)
Player relative (Self)
Health
Shields
Energy
Transport slots taken
Build progress (Not applicable to SCV, normally 0-100)

Player Observations

```
obs.observation.player.player_id
obs.observation.player.minerals
obs.observation.player.vespene
obs.observation.player.food_used
obs.observation.player.food_cap
obs.observation.player.food_army
obs.observation.player.food_workers
obs.observation.player.idle_worker_count
obs.observation.player.army_count
obs.observation.player.warp_gate_count
obs.observation.player.larva_count

free_supply = food_cap - food_used
```

Enable Feature Units

```
with sc2_env.SC2Env(
    map_name="Simple64",
    players=[sc2_env.Agent(sc2_env.Race.terran),
             sc2_env.Bot(sc2_env.Race.zerg, sc2_env.Difficulty.easy)],
    agent_interface_format=features.AgentInterfaceFormat(
        feature_dimensions=features.Dimensions(screen=84,
                                               minimap=64),
        action_space=actions.ActionSpace.FEATURES,
        use_feature_units=True),
    ) as env:
```

Feature Units

- Every visible unit on screen
- Exact unit location
- Build progress
- Assigned worker count
- Ideal worker count

```
marines = [unit for unit in obs.observation.feature_units
           if unit.unit_type == units.Terran.Marine
           and unit.alliance == features.PlayerRelative.SELF]
```

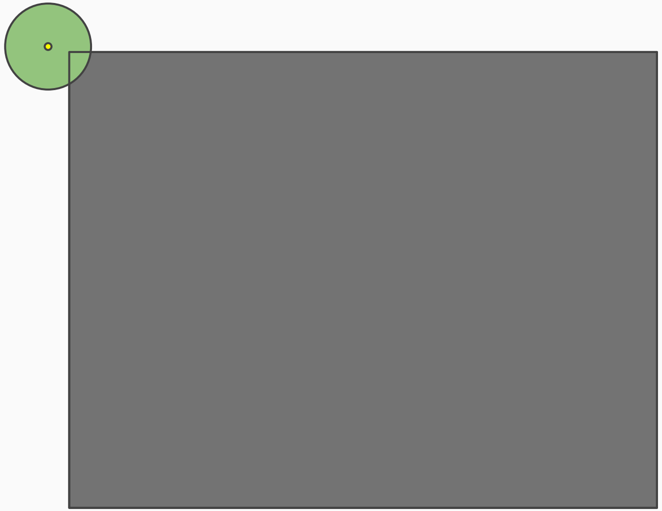
Feature Unit Properties

```
unit_type
alliance
health
shield
energy
cargo_space_taken
build_progress # 0-100
health_ratio # 0-255
shield_ratio # 0-255
energy_ratio # 0-255
display_type
owner
x
y
```

```
facing
radius
cloak
is_selected
is_blip
is_powered
mineral_contents
vespene_contents
cargo_space_max
assigned_harvesters
ideal_harvesters
weapon_cooldown
order_length
addon_unit_type # soon?
```

Feature Unit Notes

- Unit coordinates may be outside the screen since they are the centre of the unit, you will have to clip the values



Feature Unit Notes

- Unit visibility seems to match the real game, so if you are acting in the FEATURE space you may be able to move them to a location that makes them no longer visible

Machine Learning Tips

- Be sure to use categorical encoding for categorical feature layers, instead of ranges

```
[0 1 0 2 2 0 3 0]
[0 1 0 0 0 0 0 0]
[0 0 0 1 1 0 0 0]
[0 0 0 0 0 0 1 0]
```

```
0 1 2 3
```

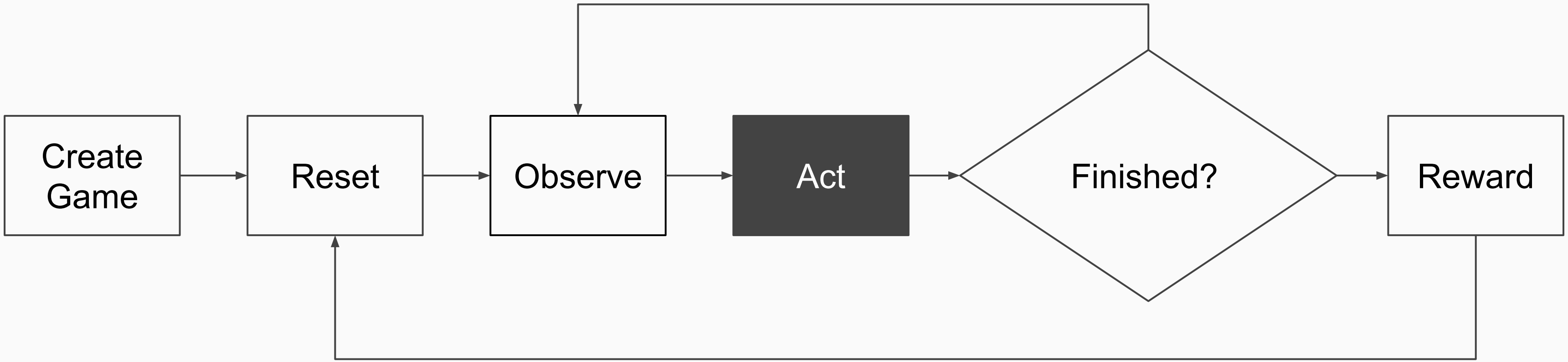
Machine Learning Tips

- Feature scaling - things like minerals and step count can number into tens of thousands, while other values may be < 10
- Consider reducing minerals to “can afford” flags
- Crop and rotate the minimap so every game is from the same perspective

Machine Learning Tips

- Make sure the agent can tell the difference between distinct states - don’t leave out crucial information

Game Engine Flow



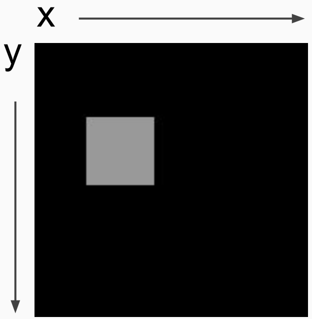
Move the Camera

Get the camera position:

```
camera_ys, camera_xs = (obs.observation.feature_minimap.camera == 1).nonzero()
camera_y = camera_ys.mean()
camera_x = camera_xs.mean()
camera_size = camera_y.max - camera_y.min # assuming square resolution
```

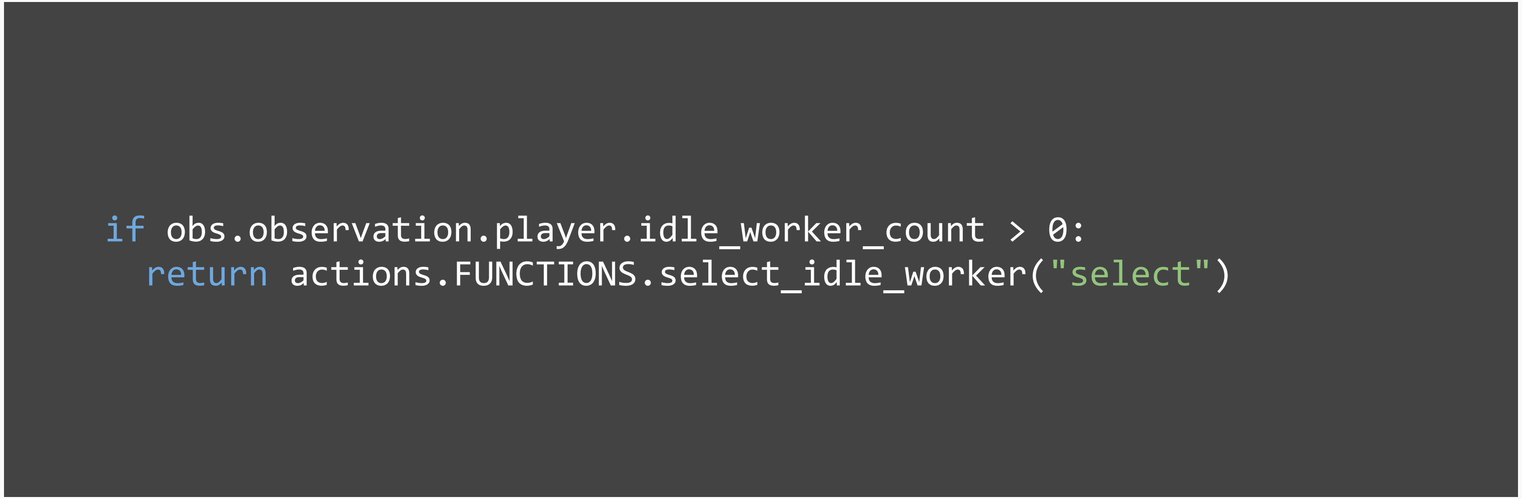
Set the camera position:

```
return actions.FUNCTIONS.move_camera((x, y)) # (x, y) is a tuple
```



- Due to rounding errors (or something) the camera position you receive may not match what you send in
- The x and y coordinates must be within the minimap size, e.g. 0-63
- Top-left is (0, 0)

Select an Idle Worker



```
if obs.observation.player.idle_worker_count > 0:
  return actions.FUNCTIONS.select_idle_worker("select")
```

Selecting idle workers will move the screen

You can also “select_all”

Build a Barracks

```
if (actions.FUNCTIONS.Build_Barracks_screen.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Build_Barracks_screen("now", (x, y))
```

You can help your agent by preventing it from building in the mineral line

For Protoss or Zerg agents you can help by limiting to power or creep

Build a Barracks

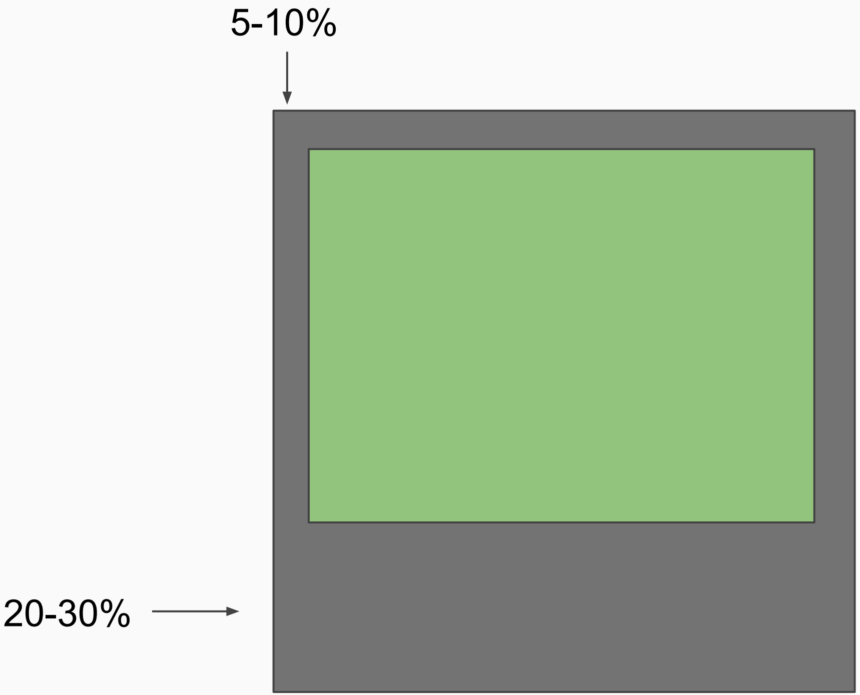
```
if (actions.FUNCTIONS.Build_Barracks_screen.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Build_Barracks_screen("now", (x, y))
```

You can help your agent by preventing it from building in the mineral line

For Protoss or Zerg agents you can help by limiting to power or creep

Building Locations

It’s best to add a margin to the left, top, and right, and a larger margin to the bottom



5-10%

20-30%

Autocast Repair

```
if (actions.FUNCTIONS.Effect_Repair_autocast.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Effect_Repair_autocast()
```

Harvest Minerals

```
if (actions.FUNCTIONS.Harvest_Gather_screen.id in
    obs.observation.available_actions):
  minerals = [unit for unit in self.obs.observation.feature_units
              if unit.unit_type in MINERAL_UNIT_TYPES]
  if len(minerals) > 0:
    mineral = random.choice(minerals)

    return actions.FUNCTIONS.Harvest_Gather_screen(
        "queued", (mineral.x, mineral.y))
```

I have not had a lot of success with queued mineral harvesting

Mineral Unit Types

```
units.Neutral.BattleStationMineralField
units.Neutral.BattleStationMineralField750
units.Neutral.LabMineralField
units.Neutral.LabMineralField750
units.Neutral.MineralField
units.Neutral.MineralField750
units.Neutral.PurifierMineralField
units.Neutral.PurifierMineralField750
units.Neutral.PurifierRichMineralField
units.Neutral.PurifierRichMineralField750
units.Neutral.RichMineralField
units.Neutral.RichMineralField750
```

Vespene Unit Types

```
units.Neutral.ProtossVespeneGeyser
units.Neutral.PurifierVespeneGeyser
units.Neutral.RichVespeneGeyser
units.Neutral.ShakurasVespeneGeyser
units.Neutral.SpacePlatformGeyser
units.Neutral.VespeneGeyser
```

Selecting all Barracks

```
barracks = [unit for unit in self.obs.observation.feature_units
            if unit.unit_type == units.Terran.Barracks
            and unit.alliance == features.PlayerRelative.SELF]
if len(barracks) > 0:
  barrack = random.choice(barracks)

  return actions.FUNCTIONS.select_point(
      "select_all_type", (barrack.x, barrack.y))
```

Remember to clip the x and y coordinates to fit the screen resolution (e.g. 0-83)

Barrack is not the singular for barracks, I know

Set Rally Point

```
if (actions.FUNCTIONS.Rally_Units_screen.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Rally_Units_screen("now", (x, y))
```

You can also rally workers

You can also use the minimap

Add Barracks to Control Group

```
return actions.FUNCTIONS.select_control_group("append", 0)
```

You can also set and recall

I have had issues with this

Train a Marine

```
if (actions.FUNCTIONS.Train_Marine_quick.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Train_Marine_quick("now")
```

There seems to be a bug currently that stops the correct distribution of build orders

In my experience when you have multiple units selected, all commands apply to all units even if normally they would not (e.g. SCVs and buildings)

Upgrade to Orbital Command

```
if (actions.FUNCTIONS.Morph_OrbitalCommand_quick.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Morph_OrbitalCommand_quick("now")
```

Scan

```
if (actions.FUNCTIONS.Effect_Scan_screen.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Effect_Scan_screen("now", (x, y))
```

You can also scan the minimap

Build Tech Lab

```
if (actions.FUNCTIONS.Build_TechLab_quick.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Build_TechLab_quick("now")
```

There is currently a bug that only allows “Build_Techlab_screen”, so you need to lift with “Lift_Barracks_quick” and then use “Build_Techlab_screen” with coordinates

Action Grouping

- By default common actions are grouped, e.g. “burrow” for Zerg units has an individual command for each unit, but when they are grouped you only need to issue one command
- You can disable this by setting `hide_specific_actions=False` in your agent interface format

Research Stim

```
if (actions.FUNCTIONS.Research_Stimpack_quick.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Research_Stimpack_quick("now")
```

It’s hard to know if your research is done since research build queues don’t exist (yet?), you may have to internally track how long it has been in progress, and check again if the research action is available (and you have enough resources)

Stim

```
if (actions.FUNCTIONS.Effect_Stim_quick.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Effect_Stim_quick("now")
```

Attack

```
if (actions.FUNCTIONS.Attack_minimap.id in
    obs.observation.available_actions):
  return actions.FUNCTIONS.Attack_minimap("now", (x, y))
```

You can also attack the screen

You can also move units without attacking

Machine Learning Tips

- Consider scripted action sequences
- Consider scripted building locations
- Consider scripted build orders
- Attacking is simpler if you choose the closest enemy location
- Consider automated supply management
- Consider automated worker management

Machine Learning Tips

- Try limiting actions only to those that are valid in the current state
- If limiting actions, remember to exclude those actions when learning

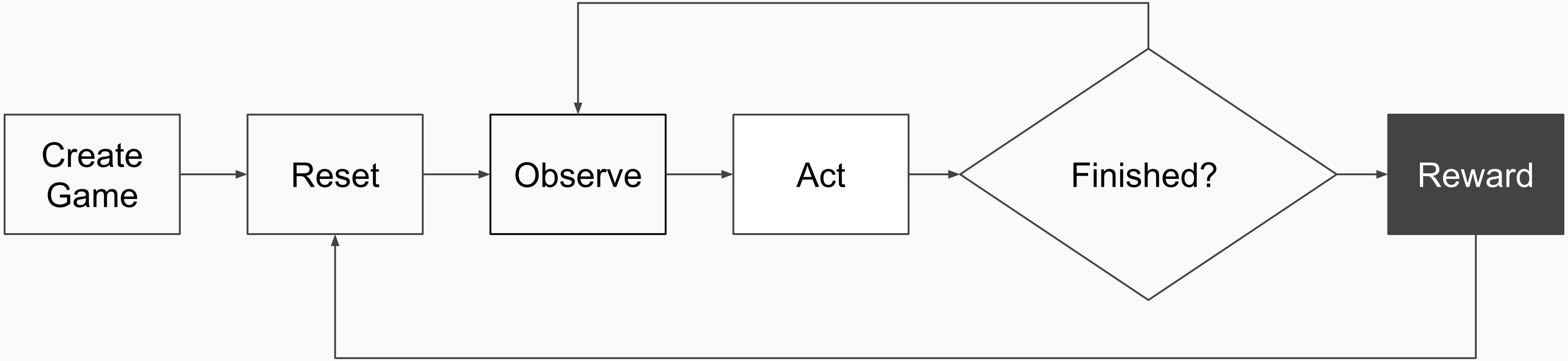
```
[-0.1,  0.0] # max is 0.0, should be -0.1
[ 0.0,  0.0] # max is 0.0
```

Machine Learning Tips

- Do not learn if the state does not change (unless it’s the last step) as actions will tend towards the highest value even if they had no impact

```
[   0.0,    -0.1,   0.1]
[  0.01,    -0.1,   0.1] # action 0
[   0.0, -0.008,   0.1] # action 1
```

Game Engine Flow



PySC2 Rewards

```
if obs.last():
  reward = obs.reward

  # train your agent
  # save your learning

  return actions.FUNCTIONS.no_op()
```

Machine Learning Tips

- You can define your own reward structure
- Be careful what you reward - you just might get it
- You might split your agent and reward for different things, for example:
    - Building units
    - Killing units
    - Finding the enemy

Limitations

- Cannot accept surrender
- Hard to track upgrades
- No research progress
- No ghosted buildings
- No build queue in multi-select
- Alerts/messages seem to be limited and unreliable

Other Questions

- Can you play against your bot?
- Does it support supervised learning?
- Can it process replays?

Windows vs Linux

- Linux is used by DeepMind, so that is most thoroughly tested
- Linux releases lag behind the latest balance changes and ladder maps
- Linux bugs take longer to get fixed
- Windows stays updated as it’s the official game
- PySC2 often lags behind the latest Windows balance changes and ladder maps

More Resources

- https://discord.gg/b2gjyHR
- https://medium.com/@skjb/build-a-zerg-bot-with-pysc2-2-0-295375d2f58e