Augmentations for BWAPI to help new users starting out

Timothy Sharples
Supervising Tutor: Prof W. Faber
University of Huddersfield

Abstract

For part of research into ways to make a simplified way of making bots for RTS games, some additions for BWAPI were designed to help new users making their first bot. These are designed for users that are first starting out with BWAPI and might struggle with some of the more complex workings of developing in a non-deterministic environment. They aren't there to be used in every project, and some users may find them restrictive after a time, however they are designed for new users who might find starting out with BWAPI in its entirety overwhelming.

Contents

1 Production Queue **5**
1.1 Constructor . . . 5
1.2 onGameStart . . . 5
1.3 Update . . . 5
1.4 AddToQueue . . . 6
1.5 RemoveFromQueue . . . 6
1.6 unitStructureStarted . . . 6
1.7 unitStructureDestroyed . . . 7
1.8 checkIfHaveInProduction . . . 7
1.9 checkHowManyInProduction . . . 7

2 Production Order **8**
2.1 getPriority . . . 8
2.2 getOrderTime . . . 8
2.3 getStatus . . . 8
2.4 setStatus . . . 8
2.5 toString . . . 9
2.6 canAfford . . . 9
2.7 isToProduceFree . . . 9
2.8 checkHasStarted . . . 9
2.9 checkHasFinished . . . 9
2.10 getToProduce . . . 10
2.11 UnitBuildingOrder Specific - setStartedUnit . . . 10
2.12 UnitBuildingOrder Specific - getStartedUnit . . . 10
2.13 UnitBuildingOrder Specific - enoughSupply . . . 10
2.14 UnitBuildingOrder Specific - checkIfStillTraining . . . 10

3 Enemy Base Tracker Manager **11**
3.1 Constructor . . . 11
3.2 onFrame . . . 11
3.3 createTrackersForAllEnemies . . . 11
3.4 getEnemyUnitBuildingList . . . 11
3.5 checkIfEnemyHas . . . 12
3.6 unitFound . . . 12
3.7 unitDestroyed . . . 12

4 MemoryUnitBuilding **13**
4.1 getUnit . . . 13
4.2 getLastKnownUnitType . . . 13
4.3 getLastKnownPosition . . . 13
4.4 isVisible . . . 13

5 Builder Manager **14**
5.1 Constructor . . . 14
5.2 getBuilders . . . 14
5.3 addJob . . . 14
5.4 cancelJob . . . 15
5.5 onFrame . . . 15
5.6 unitBuildingStarted . . . 15
5.7 unitBuildingComplete . . . 15
5.8 unitBuildingKilled . . . 15
5.9 getSpareBuilderCloseTo . . . 16

6 Squad Manager **17**
6.1 Constructor . . . 17
6.2 onFrame . . . 17
6.3 createSquad . . . 17
6.4 disbandSquad . . . 17
6.5 getAllSquads . . . 18
6.6 getSquad . . . 18
6.7 onUnitDestroy . . . 18
6.8 setRemoveDeadSquads . . . 18
6.9 isUnitAssignedToSquad . . . 18
6.10 getAllSquadNeeds . . . 18

7 Squad **19**
7.1 getID . . . 19
7.2 getTargetSquadMakeup . . . 19
7.3 getAssignedUnits . . . 19
7.4 getTargetPosition . . . 19
7.5 getTargetUnit . . . 19
7.6 getRightClickTarget . . . 19
7.7 getWasComplete . . . 20
7.8 getIsDead . . . 20
7.9 getSquadNeeds . . . 20
7.10 setSquadMoveTarget . . . 20
7.11 setAttackTarget . . . 20
7.12 setSquadRightClick . . . 20
7.13 addUnitToSquad . . . 21
7.14 removeUnitFromSquad . . . 21

8 Example Bot **22**
8.1 onStart . . . 22
8.2 onFrame . . . 22
8.3 gameProgressionUpdate . . . 22
8.4 onUnitCreate . . . 23
8.5 onUnitMorph . . . 23

8.6 onUnitComplete 23
8.7 onUnitDestroy 23
8.8 onUnitDiscover 23
8.9 getBuildTile 23

1 Production Queue

This holds all production orders that are requested of it. Orders will be sorted by priority first, then by time ordered. On each frame it will go through list of orders to find which are ready to be executed and will pass back a ready order.

Orders are added with the use of the AddToQueue methods, then when they are ready for execution will be returned in the form of one of the three types of ProductionOrder: UnitBuildingOrder, UpgradeOrder, or ResearchOrder.

REQUIRED USER BOT CALLS:

- onGameStart: Call onGameStart
- onFrame: Call update
- onUnitCreate: Call unitStructureStarted
- onUnitMorph: Call unitStructureStarted
- onUnitDestroy: Call unitDestroyed

1.1 Constructor

```
public ProductionQueue(boolean debugMessagesOn, boolean debugOnScreen)
```

- @param debugMessagesOn True if you want the console output debug messages detailing the workings of the production queue. False if you don't.
- @param debugOnScreen True if you want the in game on screen listing of the items in the production queue. False if you don't.

1.2 onGameStart

```
public void onGameStart()
```

Basic setup of the ProductionQueue that needs to be run at game start. This would be in the constructor but due to the way that BWAPI initialises, some things should be done on game start, after the StarCraft engine has setup.

1.3 Update

```
public ProductionOrder Update()
```

Method must be called in the users bots onFrame method. Checks are run to see if an order is ready. Any that are ready for execution will be returned for the user to execute how they want.

- @return ProductionOrder that is ready for execution. NULL if none are ready.

1.4 AddToQueue

```
public void AddToQueue(UnitType toBuild, int priority,
boolean autoBuildPrereq)

public void AddToQueue(TechType toBuild, int priority,
boolean autoBuildPrereq)

public void AddToQueue(UpgradeType toBuild, int priority,
boolean autoBuildPrereq)
```

Three overloaded methods that add an item to the production queue. Depending on the type of the toBuild object passed in will depend on what type o production order is created for the list. The parameter autoBuildPrereq determines whether any not completed prerequisites will automatically be added to the queue. In the versions of the method call that do not have this option, it is automatically taken to be true.

- @param toBuild The Unit, Building, Research or Upgrade wanted
- @param priority The Priority level of this order. The Higher the number, the higher the priority.
- @param autoBuildPrereq True will automatically build all prerequisite structures not currently owned. False will only add this order to the queue.

1.5 RemoveFromQueue

```
public void RemoveFromQueue(UnitType toRemove, boolean removeAll)

public void RemoveFromQueue(TechType toRemove)

public void RemoveFromQueue(UpgradeType toRemove)
```

Three overloaded methods that remove an item or items from the production queue. Due to it not being possible to have multiple instances of the same tech or upgrade in the queue, only the unit or building removal method has the option to remove all.

- @param toRemove The Unit or Building to be cancelled
- @param removeAll True will remove all of that type from the queue. False will remove only the first one found starting with the lowest priority.

1.6 unitStructureStarted

```
public void unitStructureStarted(Unit startedUnit)
```

This method must be called by the users bot on both the onUnitCreate trigger from the base API and the onUnitMorph trigger. This informs the queue that a unit or building has been started and the order will store the reference to that unit or building to be able to check if it has completed.

- @param startedUnit The started unit

1.7 unitStructureDestroyed

```
public void unitStructureDestroyed(Unit destroyedBuilding)
```

This method must be called by the users bot on the onUnitDestroy trigger from the base API. This informs the queue that a building was destroyed. It will then go through and find out if that was one of the ones that was either under construction for an order, or if it was producing something for an order. If it was then it will set the associated order to commissioned again.

- @param destroyedBuilding The destroyed unit

1.8 checkIfHaveInProduction

```
public boolean checkIfHaveInProduction(UnitType toCheck)
```

```
public boolean checkIfHaveInProduction(TechType toCheck)
```

```
public boolean checkIfHaveInProduction(UpgradeType toCheck, int level)
```

Three overloaded methods that runs a check to see if the player currently has one of this item in the production queue.

- @param toCheck The item to check
- @return True if one is in the queue. False if it isn't.

1.9 checkHowManyInProduction

```
public int checkHowManyInProduction(UnitType toCheck)
```

Runs a check of the production queue to see how many of a certain unit or building are currently in the queue

- @param toCheck The unit type to check how many are there
- @return The integer value of how many are in the queue

2 Production Order

The abstract base class for the three types of production order. Used with the production queue, the user will need to be able to take one of these orders and interpret it to find out what the bot should create after it has been given to them by the update method of the production manager.

2.1 getPriority

```
public int getPriority()
```

Gets the priority of the production order

- @return Int Priority : Higher the number the higher the priority.

2.2 getOrderTime

```
public int getOrderTime()
```

Gets the time that the order was made

- @return Int orderTime : The number of frames that had passed when the order was made.

2.3 getStatus

```
public OrderStatus getStatus()
```

Gets the status of the order

- @return OrderStatus enum CurrentStatus : The current status of the order
- Order Status's
  - commissioned - on the queue but not yet acted upon. Not enough resources or not a priority
  - ordered - Enough Resources have been gathered, there is a free to produce unit and the order has been passed back to the users bot to begin production.
  - started - work on the has begun.
  - aborted - when a job has been cancelled but has not been cleared off the queue yet.

2.4 setStatus

```
public void setStatus(OrderStatus statusIn)
```

Sets the status of the order

- @param statusIn The status to set the order to

2.5 toString

```
public abstract String toString();
```

Gets the string representation of the order, different for unit/building, research or upgrade but the general form is:

Create: Unit/Building - , Priority - , Time - , Status -

- @return The string representation of the order

2.6 canAfford

```
public abstract boolean canAfford();
```

Run a check to see if the current player can afford to purchase what the order s going to create

- @return Boolean : true means the player can afford it, false means they can't

2.7 isToProduceFree

```
public abstract boolean isToProduceFree();
```

Run a check to see whether there is a building that can produce this item and if it is idle

- @return True if there is an idle building of the type needed. False if there isn't.

2.8 checkHasStarted

```
public abstract boolean checkHasStarted();
```

Run a check to see if order has been started. Only to be run if the order is 'ordered'. Does nothing for UnitBuildingOrders. If it is started then the status in the order is set to started.

- @return Boolean. True if the order has started. False if it hasn't

2.9 checkHasFinished

```
public abstract boolean checkHasFinished();
```

Run a check to see if the order has finished. Only to be run if the order is 'started'. It it is finished then the status in the order is set to finished.

- @return Boolean. True if the order is finished. False if it isn't.

2.10 getToProduce

```
public UnitType getToProduce()
```

```
public UpgradeType getToProduce()
```

```
public TechType getToProduce()
```

Different for each specific type of Production order, and only accessible by parsing the Production Order to the correct subtype, this will return the item to be produced by the order.

- @return UnitType. The type of unit or building to be produced.
- @return UpgradeType. The upgrade to be produced.
- @return TechType. The research to be produced.

2.11 UnitBuildingOrder Specific - setStartedUnit

```
public void setStartedUnit(Unit unitIn)
```

Automatically called by Production Queue unitStructureStarted method. Set the unit that is being produced in the order. This is later used to check if it is finished.

- @param unitIn The unit that has just started for this order.

2.12 UnitBuildingOrder Specific - getStartedUnit

```
public Unit getStartedUnit()
```

Get the unit that is being produced or built.

- @return The unit that is being produced or built.

2.13 UnitBuildingOrder Specific - enoughSupply

```
public boolean enoughSupply()
```

Checks to see if the bot currently has enough supply for the unit or building to be produced.

- @return True if there is enough supply. False if not.

2.14 UnitBuildingOrder Specific - checkIfStillTraining

```
public boolean enoughSupply()
```

Check to see if the unit that was being built still exists. This will be false if the building creating it was destroyed or the unit was cancelled manually.

- @return True if the unit is still being built. False if it isn't

3 Enemy Base Tracker Manager

Manager that holds all of the different Enemy Base Trackers. Then allows for accessing each trackers records while keeping them up to date. Users should not need to interact with individual EnemyBaseTracker objects.

REQUIRED USER BOT CALLS:

- onGameStart - Call Constructor. Then call createTrackersForAllEnemies
- onFrame - Call onFrame
- onUnitFound - Call foundUnit
- onUnitDestroy - Call unitDestroyed

3.1 Constructor

```
public EnemyBaseTrackerManager(boolean visualsOn)
```

Setup of the EnemyBaseTracker Manager. Needs to be run before any other method.

- @param visualsOn True if the on screen visual representations of unit locations are wanted

3.2 onFrame

```
public void onFrame()
```

Must be run in the users bots onFrame method. Runs the Update method for each EnemyTracker. Allowing each record to stay up to date with the most recent movement information and unit status.

3.3 createTrackersForAllEnemies

```
public void createTrackersForAllEnemies()
```

Creates new Trackers for each enemy in the game. Must be run in the game start method.

3.4 getEnemyUnitBuildingList

```
public ArrayList<MemoryUnitBuilding> getEnemyUnitBuildingList(Player forEnemy
```

Gets all MemoryUnitBuilding records from the tracker for the requested memory.

- @param forEnemy The enemy you want the records for.
- @return ArrayList of MemoryUnitBuildings: All current records for that enemy.

3.5 checkIfEnemyHas

```
public boolean checkIfEnemyHas(UnitType type, Player forEnemy)
```

Checks to see if a given enemy has a unit or building

- @param type The unit or building to check for
- @param forEnemy The enemy to check for
- @return True if that enemy has been discovered to have that unit or building. False if they don't, or it hasn't been discovered that they do.

3.6 unitFound

```
public void unitFound(Unit foundUnit)
```

Must be run in the users onUnitFound method. Tell the manager than a unit has been found. Manager will discern which player its from and add it to the records if pertinent.

- @param foundUnit The unit found.

3.7 unitDestroyed

```
public void unitDestroyed(Unit destroyedUnit)
```

Must be run in the users onUnitDestroy method. Tell the manager than a unit has been destroyed. Manager will discern which player its from and amend it in the records if pertinent.

- @param destroyedUnit The unit destroyed.

4 MemoryUnitBuilding

Storage Class for details about an enemy unit, used with the EnemyBasedTracker Manager. When a unit is no longer visible, it's details can't be accessed so a few pertinent details are stored along with the unit class. Not creatable by user but will be given a list of these objects when requesting all details on an enemy from the EnemyBaseTracker Manager.

4.1 getUnit

```
public Unit getUnit()
```

Gets the unit that this Memory contains. Interrogation of the Unit class will return null if tried on a not visible unit.

- @return the contained memory unit

4.2 getLastKnownUnitType

```
public UnitType getLastKnownUnitType()
```

Gets the last known unit type of this unit. Changes will happen most often with Zerg morphing units, but also interrogation of the Unit class will return null if tried on a not visible unit.

- @return The last known Unit type of the unit.

4.3 getLastKnownPosition

```
public TilePosition getLastKnownPosition()
```

Gets the last seen location of the unit when it disappeared into the fog of war.

- @return Tile position of last known location.

4.4 isVisible

```
public boolean isVisible()
```

Returns whether the unit is currently visible

- @return True if the unit is visible and the details are now accessible. False if not.

5 Builder Manager

Manages the constructing of buildings and looks after keeping idle workers mining minerals. You can pass in jobs of what to build and where to build it and the manager will get a builder for it and set it to creating that building, and the manager will look after replacing builders if they are destroyed. Idle builders will be sent mining, this includes when the builder is first produced, if the building it was creating was completed or if it was moved and now finished its movement.

REQUIRED USER BOT CALLS:

- onGameStart: call Constructor
- onFrame: call onFrame
- onUnitCreate: call unitBuildingStarted
- onUnitMorph: call unitBuildingStarted
- onUnitComplete: call unitBuildingComplete
- onUnitDestroy: call unitBuildingKilled

5.1 Constructor

```
public BuilderManager(boolean debugMessagesOn)
```

Constructor. Initializes the two Arrays of Jobs and Builders.

- @param debugMessagesOn True if you want the console output debug messages detailing the workings of the production queue. False if you don't.

5.2 getBuilders

```
public ArrayList<Unit> getBuilders()
```

Gets the list of all builders

- @return ArrayList of all builder units

5.3 addJob

```
public void addJob(UnitType toBuild, TilePosition position)
```

Tells the manager to get a builder and start constructing the required building at the location. Assumption is made that the player has the resources to do so when this method is called.

- @param toBuild The building to construct
- @param position The location to build it at

5.4 cancelJob

```
public void cancelJob(UnitType toCancel, TilePosition position)
```

Cancels a previously requested job.

- @param toCancel The type of building to cancel
- @param position The location it as to be built

5.5 onFrame

```
public void onFrame()
```

Must be called in the users onFrame method. Catches any idle workers and sends them mining to their closest mineral source.

5.6 unitBuildingStarted

```
public void unitBuildingStarted(Unit started)
```

Must be called in the users onUnitCreate and onUnitMorph methods. Call to let the manager know that a unit has been started. The manager will then ascertain if it is relevant to the builder manager or not. If it is relevant then the construction job will be marked as started.

- @param started the unit that has been started

5.7 unitBuildingComplete

```
public void unitBuildingComplete(Unit completed)
```

Must be called during the users onUnitComplete method. Call to let the Manager know that a unit has been completed. The manager will ascertain if it is relevant. If it is relevant then the manager will either add the builder to the builder list. Or will mark the job as complete and remove it from the list.

- @param completed The completed unit

5.8 unitBuildingKilled

```
public void unitBuildingKilled(Unit killedUnitBuilding)
```

Must be called during the users onUnitDestroy method. Call to inform the manager that a unit or building has been destroyed. The manager will then ascertain if it is relevant to it. If it was a unit relevant to the manager then the manager will either retrieve a new builder or will cancel the job if it was the building destroyed.

- @param killedUnitBuilding The unit destroyed.

5.9 getSpareBuilderCloseTo

```
public Unit getSpareBuilderCloseTo(Position closeTo)
```

Retrieves the closest builder that is only mining minerals

- param closeTo What you are getting the closest builder to
- @return the builder

6 Squad Manager

Allows for the creation of squads with unit type lists for what they should consist of. The squad will then keep a list of what units are assigned to it and can compare what it has to what its supposed to have. The squad allows for groups of units to referred to easily on mass to give targets or movement direction.

REQUIRED USER BOT CALLS:

- onGameStart : constructor
- onFrame : onFrame
- onUnitDestroy : onUnitDestroy

6.1 Constructor

```
public SquadManager(boolean debugMessagesOn)
```

- @param debugMessagesOn True if you want the console output debug messages detailing the workings of the production queue. False if you don't.

6.2 onFrame

```
public void onFrame()
```

Must be called in the users onFrame method. Each frame checks to see if any squads are now dead and will remove them from the list if the flag RemoveDeadSquads is true.

6.3 createSquad

```
public int createSquad(HashMap<UnitType, Integer> squadMakeup)
```

Create a new squad with the desired unit make up

- @param squadMakeup The Map of units that are to be created. HashMap of Unit Types to the integer count of them
- @return the unique squad ID

6.4 disbandSquad

```
public void disbandSquad(int squadID)
```

Disbands the squad by removing it from the list of squads. Units that were in the squad will no longer return true to is unit in a squad

- @param squadID The unique squad ID to be disbanded

6.5 getAllSquads

```
public ArrayList<Squad> getAllSquads()
```

Gets all the squads

6.6 getSquad

```
public Squad getSquad(int squadID)
```

Gets the squad with the requested ID

- @param squadID the ID of the squad requested
- @return The requested Squad if it exists. Null if it doesn't

6.7 onUnitDestroy

```
public void onUnitDestroy(Unit destroyedUnit)
```

Must be called in the users onUnitDestroy method. Lets the manager know a unit has been destroyed. This will let each squad check to see if it was one of there units destroyed.

- @param destroyedUnit The destroyed Unit

6.8 setRemoveDeadSquads

```
public void setRemoveDeadSquads(boolean setToo)
```

Set to true by default, this will remove squads from the squad list if all units in it are dead and at some point it was complete. If false then the squad will remain in the list, just with no units assigned.

- @param setToo Set the parameter to. True if want to remove the dead squads, false if want to leave them.

6.9 isUnitAssignedToSquad

```
public boolean isUnitAssignedToSquad(Unit unitToCheck)
```

Runs a check in each squad to see if the requested unit is attached to that squad

- @param unitToCheck The unit to check if it belongs to a squad
- @return True if the unit is in a squad. False if it isn't

6.10 getAllSquadNeeds

```
public HashMap<UnitType, Integer> getAllSquadNeeds()
```

Gets the needs of all squads compiled into one hashmap

- @return Hashmap of UnitType : Integer of all the squad needs

7 Squad

Holds a list of desired squad make up and a list of assigned units. Allows for mass giving of orders to all units in the squad. Used with the Squad Manager, should not be instantiated by the user.

7.1 getID

```
public int getID()
```

Get the Unique ID of the squad

- @return Unique ID number of the squad

7.2 getTargetSquadMakeup

```
public HashMap<UnitType, Integer> getTargetSquadMakeup()
```

Gets the target squad make up

- @return HashMap of Target Squad Make up

7.3 getAssignedUnits

```
public ArrayList<Unit> getAssignedUnits()
```

Gets all the units assigned to the squad

- @return ArrayList of assigned units

7.4 getTargetPosition

```
public Position getTargetPosition()
```

Gets the position that the squad was last sent to. Null if the last target was a unit or a right click target

- @return The position the squad was last sent to

7.5 getTargetUnit

```
public Unit getTargetUnit()
```

Gets the unit that the squad was last sent to attack. Null if the last target was a position or a right click target

- @return The unit the squad was last sent to attack

7.6 getRightClickTarget

```
public Unit getRightClickTarget()
```

Gets the unit that the squad was to act as if a player had right clicked on them to. Null if the last target was a position or a unit

- @return The unit that the squad was to act as if a player had right clicked on them to

7.7 getWasComplete

```
public boolean getWasComplete()
```

If the squad was ever complete at some point then this returns as true.

- @return True if all the squad goals were met. False if not.

7.8 getIsDead

```
public boolean getIsDead()
```

Returns if the squad is dead or not. A squad is considered dead if it currently has no members and was once complete.

- @return True if squad is dead. False if not

7.9 getSquadNeeds

```
public HashMap<UnitType, Integer> getSquadNeeds()
```

Gets the HashMap of the disparity between assigned units and the squad target make up.

- @return HashMap UnitType: Integer of the types of units missing and the amount that are still needed.

7.10 setSquadMoveTarget

```
public void setSquadMoveTarget(Position movePosition, boolean attackMove)
```

Sets the squad move target to a given map position, all units in the squad will move there or as close as they can get. If the attack move option is given then the units will attack move instead. To set the squad target to a unit without making it attack, use this function and pass in the units position.

- @param movePosition the position to move the squad to.
- @param attackMove Whether the units should attack enemies they encounter along the way.

7.11 setAttackTarget

```
public void setAttackTarget(Unit toAttack)
```

Set the target of the squad to a unit. They will then go and attack that unit.

- @param toAttack The unit the squad will attack

7.12 setSquadRightClick

```
public void setSquadRightClick(Unit rightClickTarget)
```

For each unit give it the command of right clicking on a target

- @param rightClickTarget The target to right click on

7.13 addUnitToSquad

public void addUnitToSquad(Unit unitToAdd)

Assign a unit to the squad

- @param unitToAdd The unit to be assigned

7.14 removeUnitFromSquad

public void removeUnitFromSquad(Unit unitToRemove)

Will Un assign a unit from the squad if it was in there. The unit will no longer receive squad commands

- @param unitToRemove The unit to remove from the squad.

8 Example Bot

This example bot has been created to display a simple way of using each of the different packages of the library. The strategy employed by the bot will not win a game, but the current game plan will show how to do most of the actions that a user will want their bot to do. In this section is a simple break down of what is being used in each method.

8.1 onStart

To begin with, the base API onStart method is called using the super.onStart() method call. This sets up the base API terrain analyser along with setting the game flag for manual input so that testing is easier. This must be called in the onStart method of a users bot, otherwise no map analysis data will be available.

Then after that, along with a few testing variables being set-up, each of the managers for the library packages are initialised in accordance with their individual instructions. Then, if required, any other methods that the packages need for set-up are called.

8.2 onFrame

First in the onFrame method is all the onFrame methods of the three non production queue packages. Then the return of the production queue update method is captured for analysis. If the return is not null, then there is a break down of what type of production order the return is, and then actions depending on what it turned out to be. Either calling the builder manager to add a building order, or by finding the relevant production building and telling it to produce the unit, research or upgrade.

Next there is the housekeeping actions of making sure that a new supply depot is added to the production queue if the supply is getting low, and a new builder is added to the queue if there isn't enough of them.

Finally the gameProgressionUpdate method is called that keeps track of where the bot is through its game plan and what actions it should be taking that don't need to be taken every frame.

8.3 gameProgressionUpdate

The game progression update tracker is set up so that there is a barrier that will track when certain game criteria are met for advancement. This barrier is different for each stage of the game plan/.

- Level 1 - 10 Complete Builders
- Level 2 - Complete Bunker, Complete Bunker Squad
- Level 3 - Complete Academy, Complete Scouting Squad
- Level 4 - Complete Defence Squad, Complete Stim Pack Research

When this criteria is met, the next stage of the bots game plan is sent to the production queue and orders are given.

8.4 onUnitCreate

Calls the production queue's unitStructureStarted method and the builder manager's unitBuildingStarted method.

8.5 onUnitMorph

Calls the production queue's unitStructureStarted method and the builder manager's unitBuildingStarted method.

8.6 onUnitComplete

Calls the builder manager's unitComplete method. //
Then for each squad that the squad manager holds, checks to see if that squad requires a unit of this type, and if it does, assigns the unit to that squad.

8.7 onUnitDestroy

Calls the production queue's unitStructureDestroyed method, the builder manager's unitBuildingKilled method, the enemy base tracker's unit destroyed method and the squad manager's onUnitDestroy method.

Then after that, checks to see if enough units are in production to fulfil all squad needs. If there isn't then it adds what's needed to the queue.

8.8 onUnitDiscover

Calls the enemy base tracker's unitFound method.

8.9 getBuildTile

This is an example method provided by http://sscaitournament.com/index.php?action=tutorial It will find the first available place to start a a building of a given type by outward searching from a point to see if there is an open place to build of that size.