PARTIE 0 : La structure du code SC2

```
import sc2
from sc2 import run_game, maps, Race, Difficulty, position, Result
from sc2.player import Bot, Computer


class BotTerran(sc2.BotAI):

    async def on_step(self, iteration):
        """
        Choose an action to do for this step in the game.

        :param iteration: in game iteration number. There are approximately
165 iteration per minute.
        :return: execute an action.
        """
        await print("ON STEP")


if __name__ == "__main__":

    # To run several instances just execute several time this code
    # ==> /!\ each instances multiply the processor needs and memory needs.
    # specify run speed at TRUE = normal speed and FAlSE = ultra fast
speed.

    # to play against a bot, the human player must be player 1 and placed
before the bot ! else it won't work
    # [Human(sc2.Race.Zerg), Bot(sc2.Race.Terran, MeatBot())]

    run_game(
        maps.get("AbyssalReefLE"),
        [
            Bot(Race.Terran, BotTerran()),
            Computer(Race.Zerg, Difficulty.Medium)
        ],
        realtime=False)  # time in second : 1800sec = 30 min
```

PARTIE 1 : Actions de bases pour le développement de l'économie

- **Sélectionner une unité**

```
# select a SCV whatever he is doing

if self.units(SCV).amount > 0:

    scv = self.units(SCV)[0]

# select a SCV if he is not doing anything

if self.units(SCV).idle.amount > 0:

    scv = self.units(SCV).idle[0]

# select a SCV if he ready (built)

if self.units(SCV).ready.amount > 0:

    scv = self.units(SCV).ready[0]
```

- **Créer une unité avec un bâtiment**

```
for command_center in self.units(COMMANDCENTER).ready.noqueue:
    if self.can_afford(SCV):
        await self.do(command_center.train(SCV))
```

- **Construire un bâtiment (avec un SCV)**

```
if self.units(SCV).amount < 0 and self.units(COMMANDCENTER).amount > 0 :

    worker = random.choice(self.units(SCV))
    command_centers = self.units(COMMANDCENTER).ready



    if self.can_afford(SUPPLYDEPOT):

        await self.do(worker.build(SUPPLYDEPOT,
near=command_centers.first)
```

ANNEXE:

- **Obtenir la liste de tous les vesper_geyser sur la carte. (LIST)**

```
self.state.vespene_geyser
```

- **Obtenir la liste de tous les vesper_geyser sur la carte et qui sont proche de 15 unité de distance (ou moins) d'un centre de commandement. (LIST)**

```
self.state.vespene_geyser.closer_than(15.0, command_center)
```

exemple complet :

```
async def build_refineries(self):
    """
    Build a refinery if we can afford it and if we have at least a worker.

    :return: execute action self.do(worker.build(REFINERY, vaspene)).
    """
    for command_center in self.units(COMMANDCENTER).ready:
        # This return all the places where there is a vaspene geyser.
        vaspenes = self.state.vespene_geyser.closer_than(15.0,
command_center)  # tells us where the geysers are.
        for vaspene in vaspenes:
            if not self.can_afford(REFINERY):
                break
            worker = self.select_build_worker(vaspene.position)
            if worker is None:
                break
            if not self.units(REFINERY).closer_than(1.0, vaspene).exists:
# if there is not a refinery that exists
                # close to that vaspene already,
                await self.do(worker.build(REFINERY, vaspene))
```

- **Obtenir la liste des unites enemies visible. (est une fonction de bot_ai.py qui exécute en fait : self.state.enemy_units) (LIST)**

```
self.state.enemy_units
```

- **Pour savoir si une unité est déjà en train de construire un batiment : (BOOL)**

```
self.already_pending(SUPPLYDEPOT)
```