Contents

Sette opp utviklingsmiljøet

Learn Python

Tutorials:

- https://www.w3schools.com/python/exercise.asp?filename=exercise_syntax1
- https://github.com/cabhishek/python-kata

Visual Studio Code – Python

https://code.visualstudio.com/docs/python/python-tutorial

- Installere Python: (VIKTIG: Ikke 3.7.x, men 3.6.x) https://www.python.org/ftp/python/3.6.6/python-3.6.6-amd64-webinstall.exe
    - Sørge for at Python-mappa er i PATH
- Installere Visual Studio Code
- Installere Visual Studio Code Python extension

Visual Studio Code – SC2

- Fra VS Code Terminal:
    - I command vindu: C:\>python
        - deretter >>>import pip
        - så >>>exit() for å komme tilbake til cmd
    - Fra cmd: C:\>python -m pip install --upgrade pip
    - Fra cmd: C:\>pip install --upgrade sc2
- Kopiere inn «Maps»-mappe inn i Starcraft II-mappa

Starcraft 2

Terran (The Human Race)

https://liquipedia.net/starcraft2/Terran_Units_(Wings_of_Liberty)

- Unit cost: Average cost
- Health: Can repair buildings and units (use workers)
- Mobility and good defense
- Supply: Supply Depots

Zerg (Buglike, creepy aliens)

https://liquipedia.net/starcraft2/Zerg_Units_(Wings_of_Liberty)

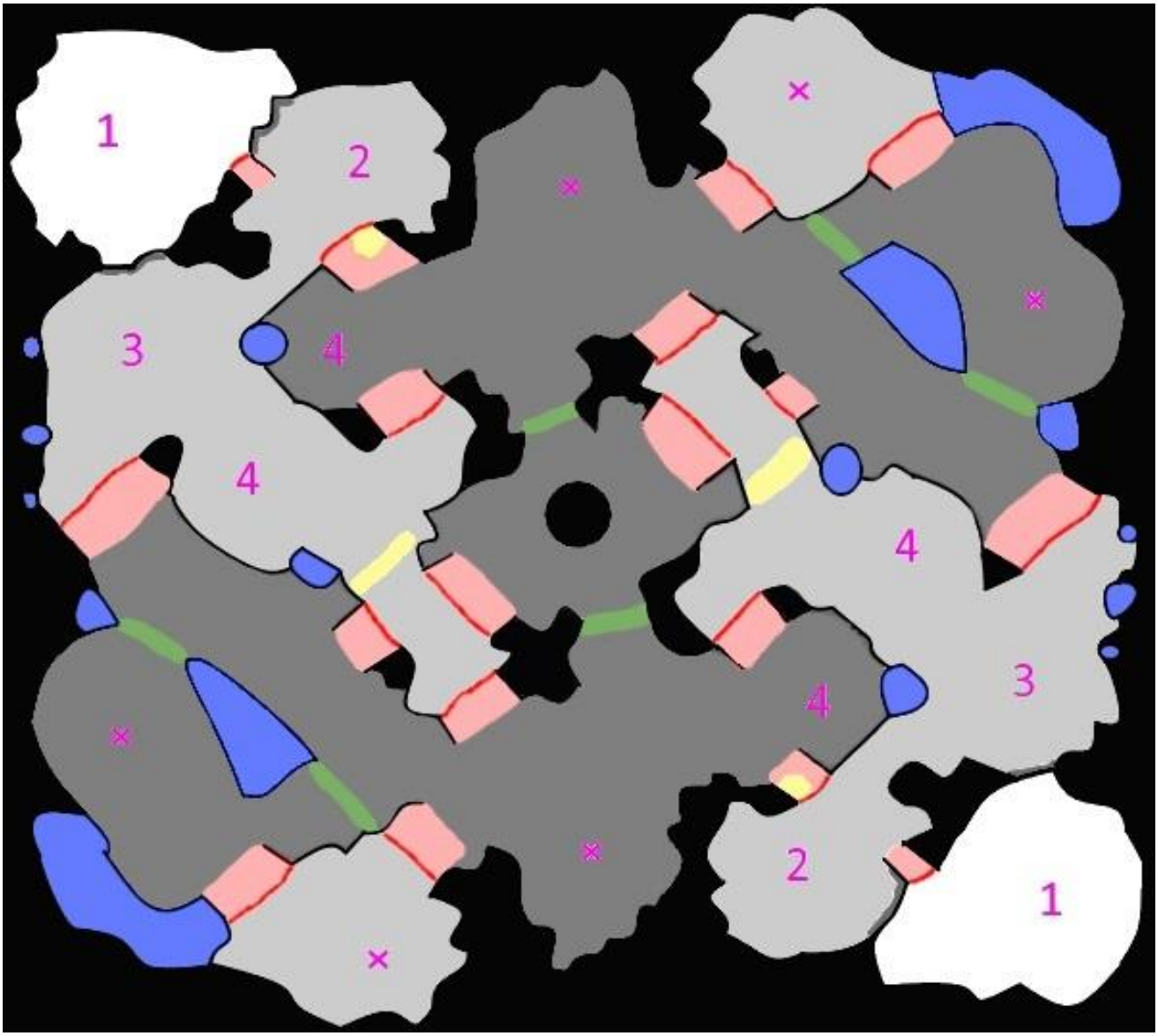
- Unit cost: Low cost, high quantity
- Drones become buildings
- Early, aggressive attack
- Health: Units and buildings regenerate (slow)
- Supply: Overlords
- Buildings must be built on the Creep

Protoss (Ancient, religious/psychic aliens)

https://liquipedia.net/starcraft2/Protoss_Units_(Wings_of_Liberty)

- Unit cost: High cost
- Buildings must be built within range of Pylons
- Can warp in units and buildings
- Health: About half their health is a (fast) regenerative force shield
- Supply: Pylons

Abyssal Reef LE



- Pink numbers and crosses are bases.
- Red areas are ramps, the line is where highground is.
- Yellow areas are rocks.
- Green areas are grass.
- Blue areas are unreachable highground (overlord spots basically)

Utvikle mot SC2-biblioteket

Starcraft 2 API

https://github.com/Dentosal/python-sc2/wiki

https://github.com/Dentosal/python-sc2/blob/master/sc2/bot_ai.py

Super-enkel bot i SC2

Først må vi importere python-filer fra **sc2**-biblioteket

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

Deretter må vi opprette en klasse som arver fra **sc2.BotAI**.

```
class FirstBot(sc2.BotAI):
```

Innunder klassen må vi implementere den asynkrone metoden **on_step** (som blir kjørt av SC2 100+ ganger i minuttet).

```
class FirstBot(sc2.BotAI):
    async def on_step(self, iteration):
        # what to do every step
```

Hver rase har en type arbeider-enhet ( Probe,  SCV,  Drone). SC2-biblioteket har en smart asynkron metode for å fordele arbeiderne best mulig: **distribute_workers**. Denne vil sørge for at man maks har 3 arbeidere per ressurs på kartet (optimalisert).

```
class FirstBot(sc2.BotAI):
    async def on_step(self, iteration):
        # what to do every step
        await self.distribute_workers()  # in sc2/bot_ai.py
```

For å starte spillet, trenger vi å definere en metode i py-filen som skal kjøre **sc2.run_game**, hvor vi velger hvilket kart vi skal starte Starcraft 2 med – og hvilke motstandere som er valgt ut. Denne metoden er ikke en del av klassen, og havner derfor med samme tab-avstand som klassen vi opprettet – og helt på bunn.

Motstanderne legges inn i rekken, og er av typen **Bot**, **Computer** eller **Human**. Kartet man benytter må finnes i Maps-mappen til Starcraft 2, og vil begrense hvor mange motstandere man kan legge til. I dette tilfellet ønsker vi at en Bot skal spille mot en Computer. Forskjellen mellom Bot og Computer er at Bot er som en assistert Human (assistert av det Python-scriptet du lager). Computer er standard PC-motstander i spillet, som kan settes til Easy, Medium eller Hard.

Parameterne etter mostander-rekken, er hvorvidt simuleringen skal foregå i standard hastighet (**realtime**) eller så fort som PC'en klarer. (I tillegg kan man her legge til parameter for å lagre omgangen i en replay-fil.)

```
run_game(maps.get("AbyssalReefLE"), [
    Bot(Race.Protoss, FirstBot()),
    Computer(Race.Terran, Difficulty.Easy)
], realtime=True)
```

Så vi har så langt et oppsett hvor vi starter opp kampen på AbyssalReefLE-kartet, hvor motstanderne er en Protoss-Bot (scriptet med FirstBot-klassen) mot en Easy Computer som spiller med rasen Terran. Full kode så langt:

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


class FirstBot(sc2.BotAI):
    async def on_step(self, iteration):
        # what to do every step
        await self.distribute_workers()  # in sc2/bot_ai.py


run_game(maps.get("AbyssalReefLE"), [
    Bot(Race.Protoss, FirstBot()),
    Computer(Race.Terran, Difficulty.Easy)
], realtime=True)
```

Bot'en din vil tape kampen, siden det eneste den gjør er å samle mineraler – uten å bygge angrep/forsvar eller utvide økonomi.

Avansert Zerg

Del 1: Første kjørbare versjon

Først lager vi en enkel ramme å jobbe rundt, ikke helt ulik den enkle Bot'en vi lagde tidligere. Den kjører derimot i rask simulering (realtime=false).

Vi legger til alle imports vi trenger i løpet av tutorial'en, først som sist. Legg merke til at vi legger til noen globale felt for klassen: **iteration** og **hq**. Disse trenger vi en del senere. Merk at denne veilederen tar utgangspunkt i at man skal holde seg til en base, og angripe raskt.

Det første vi legger til, er å trene flere arbeidere/DRONE'r (disse utvikler seg fra LARVA, og blir ikke bygd i hovedhuset slik som med Protoss). LARVA dukker opp av seg selv fra hovedhuset – opptil tre er tilgjengelige rundt et hvert hovedhus.

```
from functools import reduce
from operator import or_
import random
import enum
import sc2
from sc2 import Race, Difficulty, run_game
from sc2.constants import *
from sc2.player import Bot, Computer
from sc2.data import race_townhalls

class Hydralisk(sc2.BotAI):
    async def on_step(self, iteration):
        self.iteration = iteration
        if self.townhalls.exists:
            self.hq = self.townhalls.first

        await self.train_missing_workers()

    async def train_missing_workers(self):
        larvae = self.units(LARVA)
        if self.hq.assigned_harvesters < self.hq.ideal_harvesters:
            if self.can_afford(DRONE) and larvae.exists:
                larva = larvae.random
                await self.do(larva.train(DRONE))
                return


run_game(sc2.maps.get("AbyssalReefLE"), [
        Bot(Race.Zerg, Hydralisk()),
        Computer(Race.Zerg, Difficulty.Easy)
    ], realtime=False)
```

Nå kan du kjøre dette scriptet, og starte opp Starcraft 2. Legg merke til at du ikke får bygd flere enn 14 arbeidere.

Del 2: Forberede økonomi

For å kunne lage flere enn 14 arbeidere, så må vi øke Supply. Dette gjøres ved å trene frem OVERLORD – **train_overlord**.

```
    async def train_overlord(self):
        larvae = self.units(LARVA)
        if self.supply_left < 2:
            if self.can_afford(OVERLORD) and larvae.exists:
                await self.do(larvae.random.train(OVERLORD))
                return
```

Å bygge HYDRALISK krever Vespene Gas i tillegg til mineraler. For å få tak i Vespene Gas, må vi bygge EXTRACTOR over en Vespene Geyser. Dette gjør vi via vår **build_extractor**.

```
    async def build_extractor(self):
        if self.units(EXTRACTOR).amount < 2 and not self.already_pending(EXTRACTOR):
            if self.can_afford(EXTRACTOR):
                drone = self.workers.random
                target = self.state.vespene_geyser.closest_to(drone.position)
                await self.do(drone.build(EXTRACTOR, target))
```

Det holder ikke å bygge EXTRACTOR for å få tak i gass. I tillegg må vi sette noen av DRONE'ne til å hente gassen.

```
    async def assign_workers_to_gas(self):
        for a in self.units(EXTRACTOR):
            if a.assigned_harvesters < a.ideal_harvesters:
                w = self.workers.closer_than(20, a)
                if w.exists:
                    await self.do(w.random.gather(a))
```

Så kaller vi på de nye metodene vi har opprettet (setter de over vår gamle linje som kaller på train_missing_worker).

```
        await self.train_overlord()
        await self.build_extractor()
        await self.train_missing_workers()
        await self.assign_workers_to_gas()
```

Når du nå kjører scriptet ditt i Starcraft 2, så vil du se at du lager nok DRONE'r til å dekke behovet på mineralene og gassen du har tilgjengelig umiddelbart rundt startbasen din.

Del 3: De første enhetene

Nå ønsker vi lage et enkelt forsvar. For å lage de enkleste angrepsstyrkene til Zerg, så må vi først bygge et SPAWNINGPOOL.

```
    async def build_spawningpool(self):
        if not (self.units(SPAWNINGPOOL).exists or self.already_pending(SPAWNINGPOOL)):
            if self.can_afford(SPAWNINGPOOL):
                await self.build(SPAWNINGPOOL, near=self.hq)
```

Deretter kan vi trene noen ZERGLING'er (som nok en gang vokser ut av LARVA). Riktignok ønsker vi kun å trene disse mens vi venter på HYDRALISK'ene (som er hovedstrategien til denne Bot'en), og i tillegg sørge for å ikke lage for mange.

```
    async def train_zerglings(self):
        if self.units(ZERGLING).amount < 20 and self.minerals > 1000:
            larvae = self.units(LARVA)
            if larvae.exists and self.can_afford(ZERGLING):
                await self.do(larvae.random.train(ZERGLING))
```

Siden vi kun har maks 3 LARVA tilgjengelige til enhver tid, så vil dette begrense hastigheten vi kan bygge antall enheter med. For å bøte på dette kan vi trene en QUEEN. Hun spytter inn ekstra LARVA, slik at vi har flere å trene enheter fra.

```
    async def train_queen(self):
        if self.units(SPAWNINGPOOL).ready.exists:
            if not self.units(QUEEN).exists and self.hq.is_ready and self.hq.noqueue:
                if self.can_afford(QUEEN):
                    await self.do(self.hq.train(QUEEN))
```

Så må vi få QUEEN til å spytte LARVA etter at hun er fremme i basen.

```
    async def queen_inject_larvae(self):
        for queen in self.units(QUEEN).idle:
            abilities = await self.get_available_abilities(queen)
            if AbilityId.EFFECT_INJECTLARVA in abilities:
                await self.do(queen(EFFECT_INJECTLARVA, self.hq))
```

Så utvider vi metodekallene.

```
        await self.train_overlord()
        await self.queen_inject_larvae()
        await self.build_spawningpool()
        await self.build_extractor()
        await self.train_missing_workers()
        await self.assign_workers_to_gas()
        await self.train_queen()
        await self.train_zerglings()
```

Hvis du kjører Starcraft 2 nå, så oppretter du ZERGLING og QUEEN etter hvert – og kan kanskje legge merke til at du tidvis har flere enn 3 LARVA rundt basen din pga. QUEEN.

Del 4: Din første Hydralisk

For å kunne lage HYDRALISK, må du først ha HYDRALISKDEN – som igjen krever at hovedbasen din oppgraderes fra HATCHERY til LAIR.

```
    async def upgrade_to_lair(self):
        if self.units(SPAWNINGPOOL).ready.exists:
            if not self.units(LAIR).exists and self.hq.noqueue:
                if self.can_afford(LAIR):
                    await self.do(self.hq.build(LAIR))
```

Når du har fått oppgradert hovedbasen, kan du bygge en HYDRALISKDEN.

```
    async def build_hydraliskden(self):
        if self.units(LAIR).ready.exists:
            if not (self.units(HYDRALISKDEN).exists or
self.already_pending(HYDRALISKDEN)):
                if self.can_afford(HYDRALISKDEN):
                    await self.build(HYDRALISKDEN, near=self.hq)
```

Med en operativ HYDRALISKDEN, kan vi endelig trene HYDRALISK.

```
    async def train_hydralisk(self):
        if self.units(HYDRALISKDEN).ready.exists:
            larvae = self.units(LARVA)
            if self.can_afford(HYDRALISK) and larvae.exists:
                await self.do(larvae.random.train(HYDRALISK))
                return
```

Så må vi sende de ut for å angripe, hvis vi har mange nok. I tillegg lager vi en metode for å finne fienden.

```
    async def attack_with_hydralisks(self):
        if self.units(HYDRALISK).amount > 10 and self.iteration % 50 == 0:
            forces = self.units(ZERGLING) | self.units(HYDRALISK)
            for unit in forces.idle:
                await self.do(unit.attack(self.select_target()))

    def select_target(self):
        if self.known_enemy_structures.exists:
            return random.choice(self.known_enemy_structures).position
        else:
            return self.enemy_start_locations[0]
```

Legg til de nødvendige metode-kallene, og kjør Starcraft 2 på nytt. Nå vil du ha en Bot som stort sett klarer å vinne over Easy Computer Zerg.

Del 5: Siste finpuss

Noen ganger så mister man hovedbasen, og da vil det være lurt å ikke bruke DRONE'ne til å hente ressurser, men i stedet gjøre et siste manisk angrep på fiendebasen med alt man har (inkludert arbeidere).

```
    async def townhall_is_gone(self):
        if not self.townhalls.exists:
            forces = self.units(ZERGLING) | self.units(HYDRALISK)
            for unit in self.units(DRONE) | self.units(QUEEN) | forces:
                await self.do(unit.attack(self.enemy_start_locations[0]))
```

Komplett kode for Hydralisk-bot'en:

```
from functools import reduce
from operator import or_
import random
import enum
import sc2
from sc2 import Race, Difficulty, run_game
from sc2.constants import *
from sc2.player import Bot, Computer
from sc2.data import race_townhalls

class Hydralisk(sc2.BotAI):
    async def on_step(self, iteration):
        self.iteration = iteration
        if self.townhalls.exists:
            self.hq = self.townhalls.first

        await self.attack_with_hydralisks()
        await self.train_overlord()
        await self.train_hydralisk()
        await self.townhall_is_gone()
        await self.queen_inject_larvae()
        await self.build_spawningpool()
        await self.upgrade_to_lair()
        await self.build_hydraliskden()
        await self.build_extractor()
        await self.train_missing_workers()
        await self.assign_workers_to_gas()
        await self.train_queen()
        await self.train_zerglings()

    async def train_overlord(self):
        larvae = self.units(LARVA)
        if self.supply_left < 2:
            if self.can_afford(OVERLORD) and larvae.exists:
                await self.do(larvae.random.train(OVERLORD))
                return

    async def train_hydralisk(self):
        if self.units(HYDRALISKDEN).ready.exists:
            larvae = self.units(LARVA)
```

```
            if self.can_afford(HYDRALISK) and larvae.exists:
                await self.do(larvae.random.train(HYDRALISK))
                return

    async def townhall_is_gone(self):
        if not self.townhalls.exists:
            forces = self.units(ZERGLING) | self.units(HYDRALISK)
            for unit in self.units(DRONE) | self.units(QUEEN) | forces:
                await self.do(unit.attack(self.enemy_start_locations[0]))

    async def queen_inject_larvae(self):
        for queen in self.units(QUEEN).idle:
            abilities = await self.get_available_abilities(queen)
            if AbilityId.EFFECT_INJECTLARVA in abilities:
                await self.do(queen(EFFECT_INJECTLARVA, self.hq))

    async def build_spawningpool(self):
        if not (self.units(SPAWNINGPOOL).exists or self.already_pending(SPAWNINGPOOL)):
            if self.can_afford(SPAWNINGPOOL):
                await self.build(SPAWNINGPOOL, near=self.hq)

    async def upgrade_to_lair(self):
        if self.units(SPAWNINGPOOL).ready.exists:
            if not self.units(LAIR).exists and self.hq.noqueue:
                if self.can_afford(LAIR):
                    await self.do(self.hq.build(LAIR))

    async def build_hydraliskden(self):
        if self.units(LAIR).ready.exists:
            if not (self.units(HYDRALISKDEN).exists or
self.already_pending(HYDRALISKDEN)):
                if self.can_afford(HYDRALISKDEN):
                    await self.build(HYDRALISKDEN, near=self.hq)

    async def build_extractor(self):
        if self.units(EXTRACTOR).amount < 2 and not self.already_pending(EXTRACTOR):
            if self.can_afford(EXTRACTOR):
                drone = self.workers.random
                target = self.state.vespene_geyser.closest_to(drone.position)
                err = await self.do(drone.build(EXTRACTOR, target))

    async def train_missing_workers(self):
        larvae = self.units(LARVA)
        if self.hq.assigned_harvesters < self.hq.ideal_harvesters:
            if self.can_afford(DRONE) and larvae.exists:
                larva = larvae.random
                await self.do(larva.train(DRONE))
                return

    async def assign_workers_to_gas(self):
        for a in self.units(EXTRACTOR):
            if a.assigned_harvesters < a.ideal_harvesters:
```

```
                w = self.workers.closer_than(20, a)
                if w.exists:
                    await self.do(w.random.gather(a))

    async def train_queen(self):
        if self.units(SPAWNINGPOOL).ready.exists:
            if not self.units(QUEEN).exists and self.hq.is_ready and self.hq.noqueue:
                if self.can_afford(QUEEN):
                    await self.do(self.hq.train(QUEEN))

    async def train_zerglings(self):
        if self.units(ZERGLING).amount < 20 and self.minerals > 1000:
            larvae = self.units(LARVA)
            if larvae.exists and self.can_afford(ZERGLING):
                await self.do(larvae.random.train(ZERGLING))


    async def attack_with_hydralisks(self):
        if self.units(HYDRALISK).amount > 10 and self.iteration % 50 == 0:
            forces = self.units(ZERGLING) | self.units(HYDRALISK)
            for unit in forces.idle:
                await self.do(unit.attack(self.select_target()))

    def select_target(self):
        if self.known_enemy_structures.exists:
            return random.choice(self.known_enemy_structures).position
        else:
            return self.enemy_start_locations[0]


run_game(sc2.maps.get("AbyssalReefLE"), [
        Bot(Race.Zerg, Hydralisk()),
        Computer(Race.Zerg, Difficulty.Easy)
    ], realtime=False)
```

Se på siste kapittel om Tweaks, for å eksempelsvis endre Difficulty.

Avansert Protoss

Begynn gjerne på del 2, og fortsett til og med del 6 på følgende tutorial:

https://pythonprogramming.net/starcraft-ii-ai-python-sc2-tutorial/

Ekstra Bot-Tweaks

Forandre Difficulty

```
run_game(sc2.maps.get("AbyssalReefLE"), [
        Bot(Race.Protoss, FirstBot()),
        Computer(Race.Terran, Difficulty.Medium)
    ], realtime=False)
```

```
run_game(sc2.maps.get("AbyssalReefLE"), [
        Bot(Race.Protoss, FirstBot()),
        Computer(Race.Terran, Difficulty.Hard)
    ], realtime=False)
```

```
run_game(sc2.maps.get("AbyssalReefLE"), [
        Bot(Race.Protoss, FirstBot()),
        Computer(Race.Terran, Difficulty.Harder)
    ], realtime=False)
```

Replay

I tillegg ønsker man mange ganger å ta vare på kampen som er spilt, for å analysere hva som gikk galt. For å gjøre dette så legger man til hvilken replay-fil man skal lagre matchen til.

```
run_game(sc2.maps.get("AbyssalReefLE"), [
        Bot(Race.Protoss, FirstBot()),
        Computer(Race.Terran, Difficulty.Harder)
    ], realtime=False, save_replay_as="PvT.SC2Replay")
```

Bot vs. Bot

En annen ting man kan prøve seg på, er å kjøre to Bot'er opp mot hverandre. Dette vil starte opp to Starcraft 2-sesjoner. Hvis du bare får svarte skjermer i stedet for en dobbel simulering, så kan du følge dette rådet:

"For some reason, fairly often when I put two of my own bots against each other, I just get black screens when the game launches and nothing happens. To fix this, I go to C:\Users\H\Documents\StarCraft II and delete variables.txt"

```
run_game(maps.get("AbyssalReefLE"), [
    Bot(Race.Protoss, FirstBot()),
    Bot(Race.Zerg, Hydralisk()),
    ], realtime=False)
```

Ferdige Bot-eksempler

For å se på andre Bot-eksempler: https://github.com/Dentosal/python-sc2/tree/master/examples

Kalle eksempel-bots fra SC2-biblioteket

Disse kan man også benytte i en match-up.

```
import sc2
from sc2 import run_game, maps, Race, Difficulty
from sc2.player import Bot, Computer
from sc2.constants import NEXUS, PROBE, PYLON, ASSIMILATOR, GATEWAY, \
 CYBERNETICSCORE, STALKER, STARGATE, VOIDRAY
import random
from examples.terran.proxy_rax import ProxyRaxBot
from examples.zerg.zerg_rush import ZergRushBot

run_game(maps.get("AbyssalReefLE"), [
    Bot(Race.Zerg, ZergRushBot()),
    Bot(Race.Terran, ProxyRaxBot()),
    ], realtime=False)
```