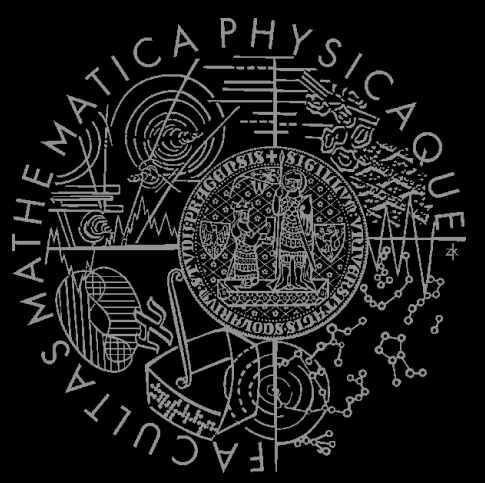
Faculty of Mathematics and Physics
Charles University in Prague
18th April 2013



UT2004 bots made easy!

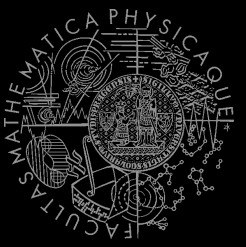
Pogamut 3

Lecture 8 – BOD, yaPOSH & DeathMatch

Warm Up!

- Fill the short test for this lessons
  - 6 minutes limit

NAIL068 Exam



- Reserved SW1 for:
  - 20. 5.2013 – 9:00-15:40
  - 23.5.2013 – 9:00-15:40
- Exam will last cca 3-4 hours (coding) + 30 minut questionnaires filling + 5 minut „informal chat“
- Do the time and date suit you?

Assignment 7 Revisited

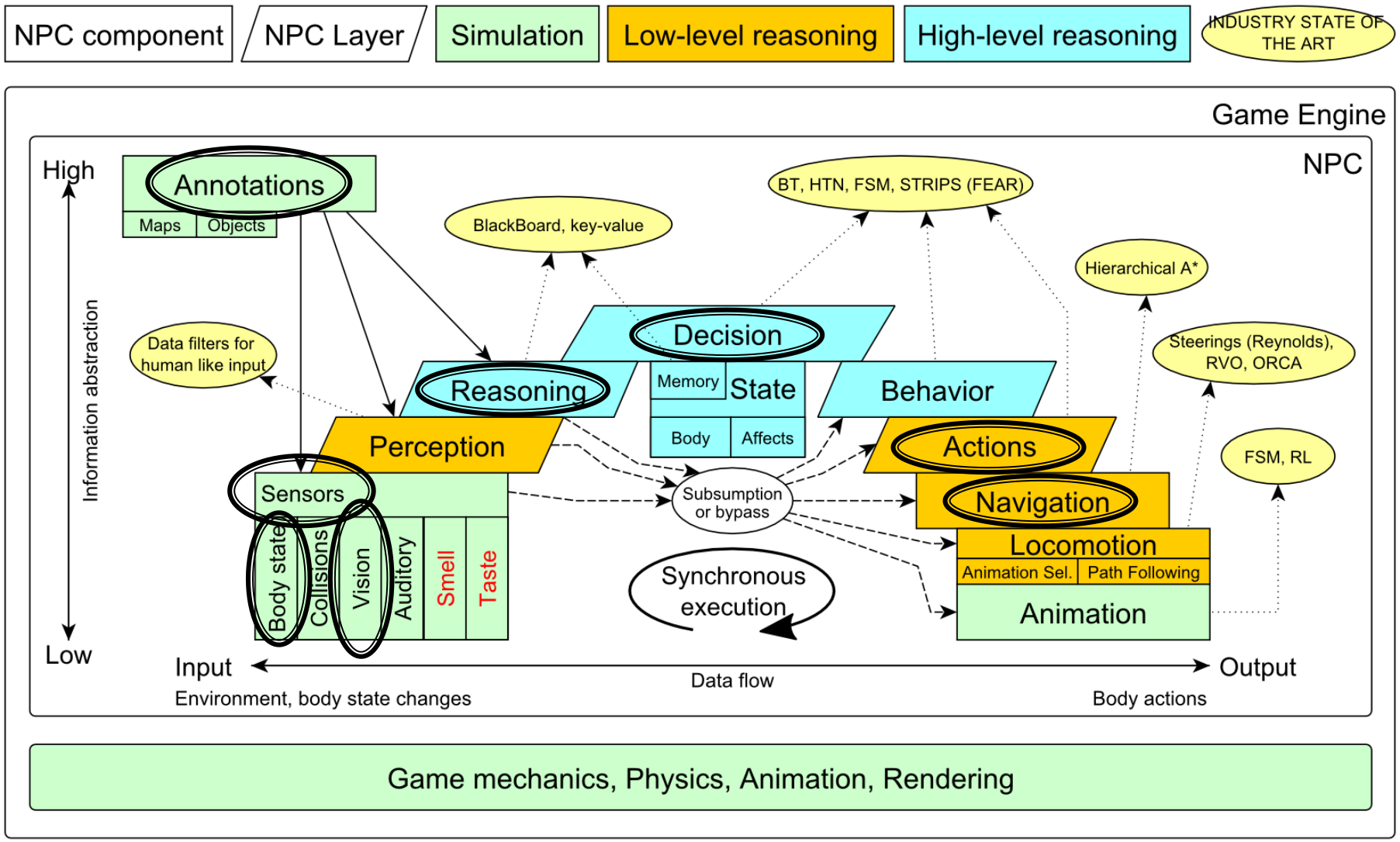
CollectorBot

```
private Item runningFor = null;
private boolean runningForPicked = false;

@EventListener(eventClass = ItemPickedUp.class)
protected void itemPickedUp(ItemPickedUp event) {
    if (runningFor == null) return;
    if (event.getId() == runningFor.getId()) {
        runningForPicked = true;
    }
}
```

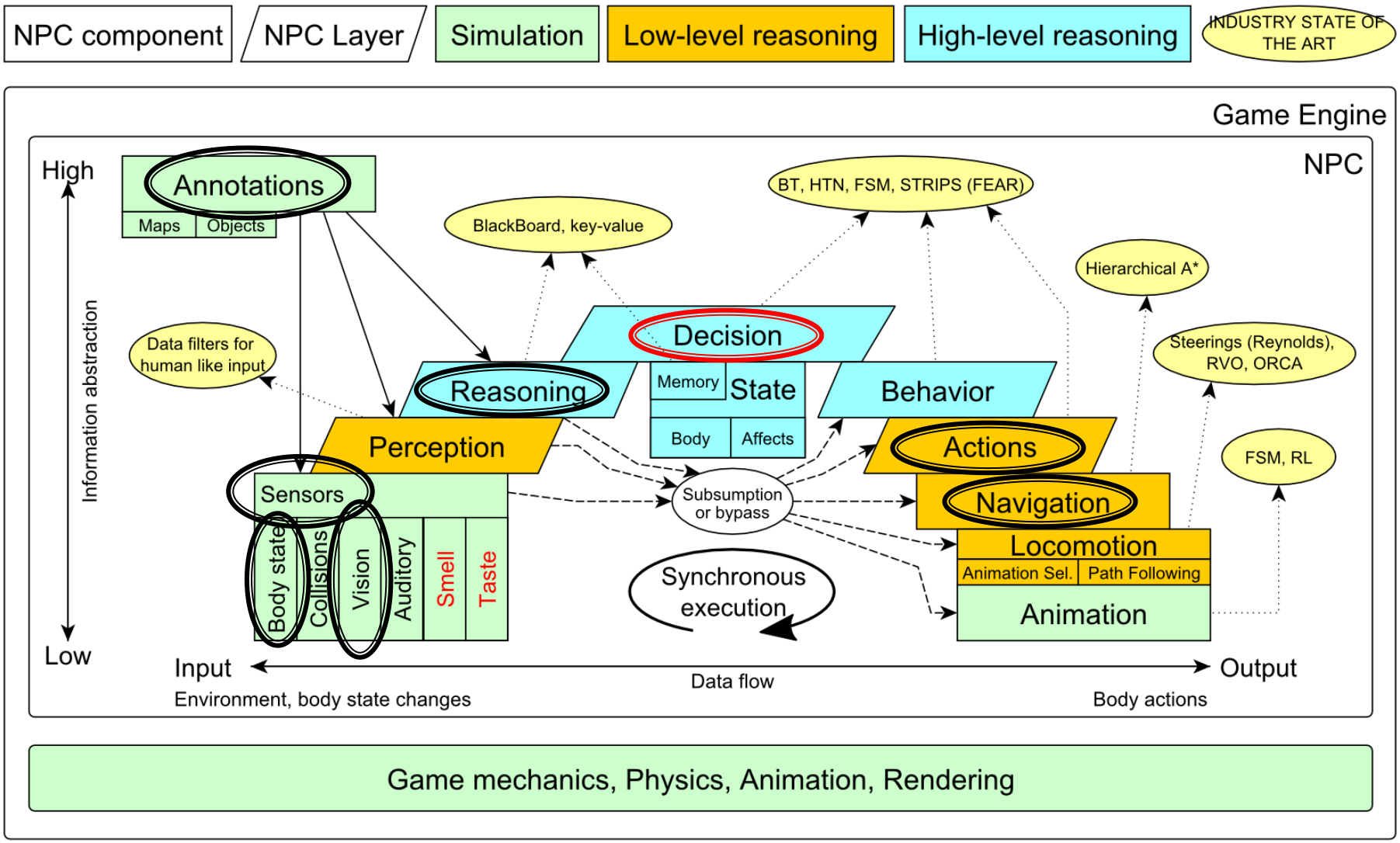
Today's menu

1. **Big Picture**
2. BOD (Behavior Oriented Design)
3. Gentle POSH introduction
4. Weapons & Shooting
5. DeathMatch Bot

Big Picture
Already covered



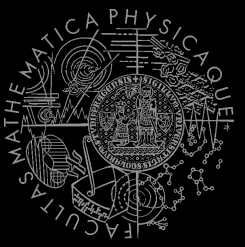
Big Picture
Today



Today's menu

1. Big Picture
2. **BOD (Behavior Oriented Design)**
3. Gentle POSH introduction
4. Weapons & Shooting
5. DeathMatch Bot

Behavior Oriented Design
Methodology



- BOD (Behavior Oriented Design)
  - A methodology for developing control of complex intelligent agents
    - virtual reality characters, humanoid robots or intelligent environments…
- Combines the advantages of Behavior-Based AI and Object Oriented Design.

- Authored by Joanna J. Bryson
  - http://www.cs.bath.ac.uk/~jjb/web/bod.html

How to think?
Intelligence by design

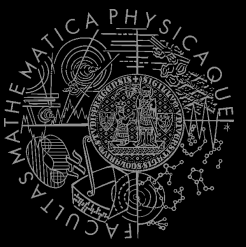
Behavior Oriented Design

by Joanna J. Bryson (UK)
http://www.cs.bath.ac.uk/~jjb/web/bod.html

1. Specify top-level decision
   a) Name the behaviors that the bot should do
   b) Identify the list of sensors that is required to perform the behavior
   c) Identify the priorities of behaviors
   d) Identify behavior switching conditions
2. Recursion on respective behaviors until primitive actions reached

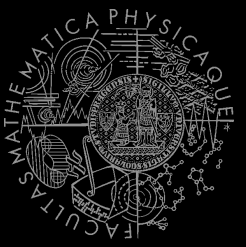
Behavior Oriented Design

BOD in human language



1. State the goal of you agent behavior
   - E.g. It will be a Deathmatch bot
2. Brainstorm what it will mean to fulfill the behavior goal
   - E.g. fight players, gather items
3. Think about conditions that should be fulfilled for the respective behaviors
   - E.g. I'll fight only when I see enemy and have proper weapon
4. Revise, revise, revise
   - Oh wait, what if I don't have the proper weapon, I should add a behavior to flee from fight and gather some weapon.
5. Pick one of the specified top level behaviors and apply recursion from point 1!
6. When you end up with sufficiently simple and clear defined sense/action – **NAME IT WELL**, implement it and test it!

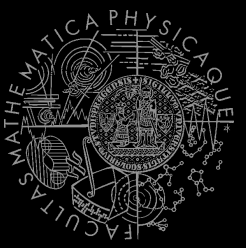
Behavior Oriented Design
Iterative Development



Recursion == Iterative development

1. Select a part of the plan to extend next.
2. Extend the agent with that implementation
   - Extend the plan, code actions and senses
   - Test and debug that code (!!!)
3. Revise the current specification.

Behavior Oriented Design
Revising BOD Specifications



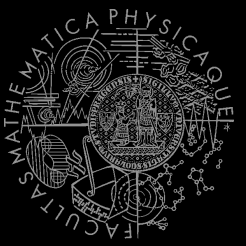
- Name the behaviors (functions) logically!
  - Good method name is better than documentation!
- Reduce code redundancy
  - Use copy past with caution or not at all!
- *Avoid Complex Conditions*
  - The shorter condition, the better the understanding
- *Avoid Too Many If-then rules at one level*
  - One level of decision making usually needs no more than 5 to 7 if-then rules, they may contain fewer..
- *When in doubt, favor simplicity.*

Practice Lesson

Outline

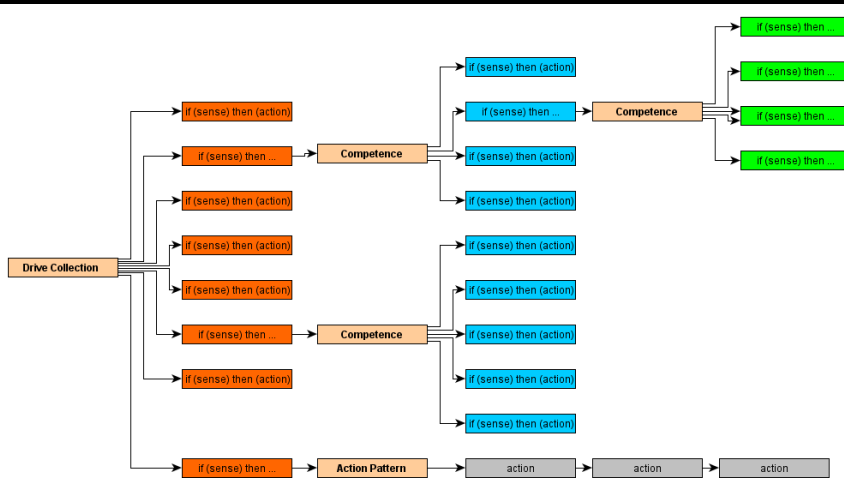
1. Big Picture
2. BOD (Behavior Oriented Design)
3. **Gentle POSH introduction**
4. Weapons & Shooting
5. DeathMatch Bot

yaPOSH
Introduction



- **yaPOSH**
  - **y**et-**a**nother **P**arallel-rooted, **O**rdered **S**lip-stack **H**ierarchical planner
- To put it simply:
  - a reactive planner working with **FIXED**, **PRE-SET** plans
- To put it even simpler:
  - a tool enabling to specify **if – then** rules with **priority** in a **tree like structure**
- Advantage:
  - Makes you think about the behavior in human terms more than the code

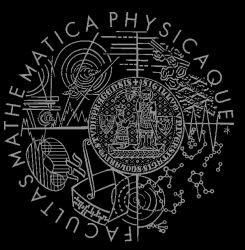
yaPOSH
Primitives



- **Actions and Senses**
  - if (sense) then (action)
- **Drive Collection (DC)**
  - First level of if-then rules
- **Competence (C)**
  - Second – Nth level of if-then rules
- **Action Patterns (AP)**
  - Specifies N actions that will be performed in a sequence

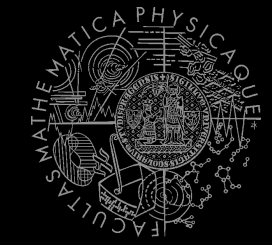
yaPOSH

Plan structure (Java glasses)

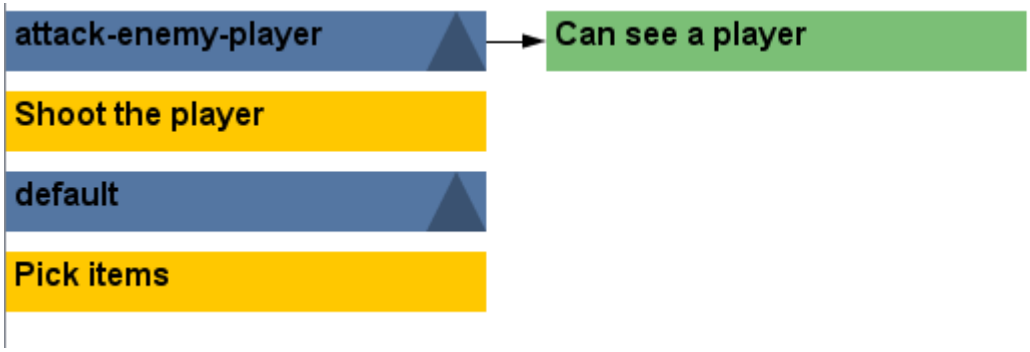



```
DriveCollection(
    1. if (sense1()) then competence1(); return;
    2. if (sense2()) then competence2(); return;
    3. if (sense3()) then action-pattern1(); return;
    4. if (sense4()) then competence3(); (
            1. if (sense5()) then action1(); return;
            2. if (sense6()) then competence4(); return;
            3. if (sense7()) then action2(); return;
            4. if (sense8()) then action-pattern(); return;
            5. return;
    )
)

ActionPattern(
    while (!action1-finished()) {action1();};
    while (!action2-finished()) {action2();};
    while (!action3-finished()) {action3();};
)
```

yaPOSH
Plan structure (the real)




```
(
        (C attack-behavior
                (elements
                        ((need-ammo cz.cuni.attackbot.PickItems))
                        ((see-no-enemy cz.cuni.attackbot.FindEnemy))
                        ((attack cz.cuni.attackbot.ShootPlayer))
                )
        )
        (DC life
                (drives
                        ((attack-enemy-player (trigger ((cz.cuni.attackbot.SeePlayer))) cz.cuni.attackbot.ShootPlayer))
                        ((default cz.cuni.attackbot.PickItems))
                )
        )
)
```

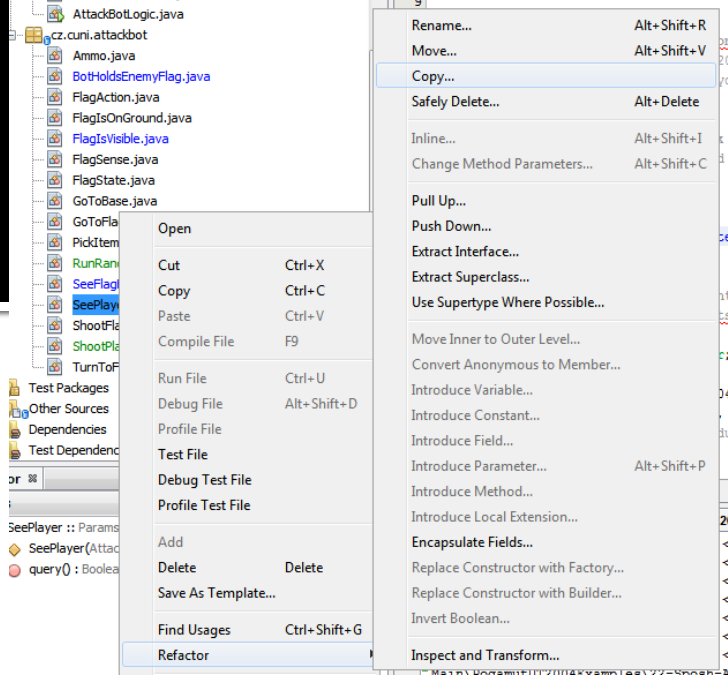


yaPOSH
Senses I

- Senses
  - Represent condition (Do I see a player?)
  - Return basic types
    - Boolean, Integer, Double, String, …
  - Can be queried either as ==, !=, >, <, <= or >=
    - E.g. cz.cuni.attackbot.FlagIsVisible false !=
  - Can be parameterized:
    - MySense extends ParamsSense<BOT_CONTEXT>
    - public void **querry**(@Param("$myParameter") String myParameter) { … }
    - cz.cuni.attackbot.MySense($myParameter = "string-value") false !=

yaPOSH

Senses II – New Sense



- How to make a new sense?
  - There are no templates yet…
- In NetBeans:
  - Right click on some existing sense,
  - Right click the Java class and select refactor and Copy it with a new name
  - Change the sense description and human readable name in the annotation before the class declaration

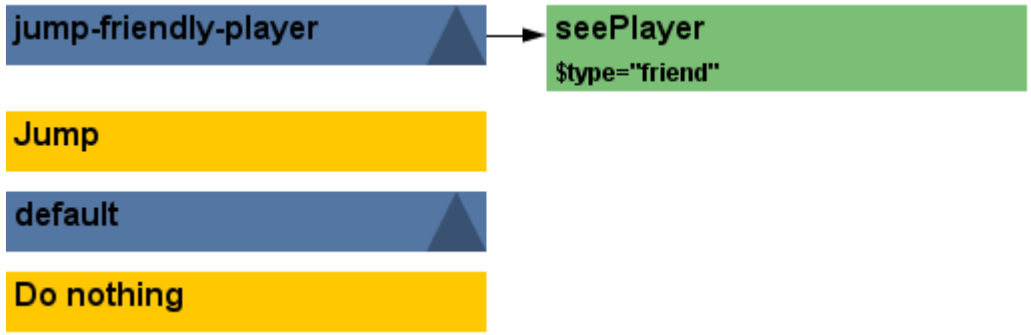
```
@PrimitiveInfo(name = "Can see a player", description = "Do I see a player?")
public class SeePlayer extends ParamsSense<AttackBotContext, Boolean> {
```

  - In POSH editor click Refresh button in the Senses editor

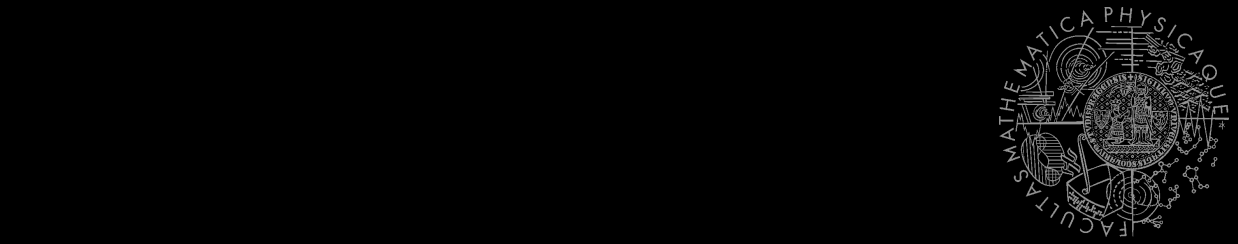
yaPOSH

Senses III – Parameterized sense example

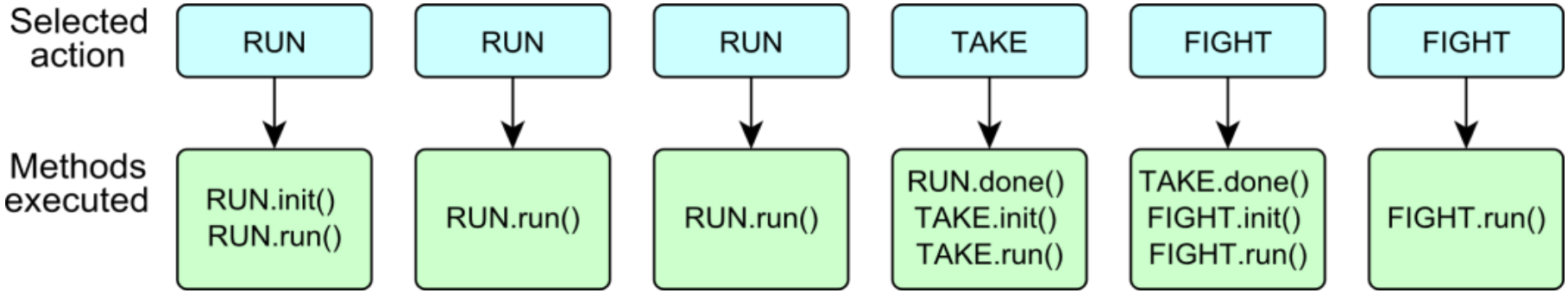

```
@PrimitiveInfo(name = "seePlayer", description = "Do I see a player?")
public class SeePlayer extends ParamsSense<AttackBotContext, Boolean> {

    public SeePlayer(AttackBotContext ctx) {
        super(ctx, Boolean.class);
    }

    public Boolean query(@Param("$type") String type) {
        if (type.contentEquals("friend"))
            return this.getCtx().getPlayers().canSeeFriends();
        else
            return this.getCtx().getPlayers().canSeeEnemies();
    }
}
```



yaPOSH
Actions I

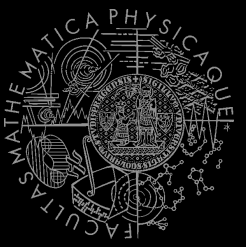


- Actions
  - Represent an action in the environment
  - Have three methods – **init(), run(), done()**



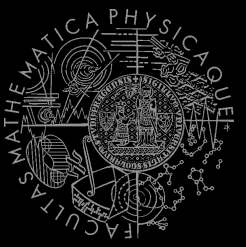
  - Can be parameterized
    - need to extend ParamsAction<BOT_CONTEXT>
    - Parameters passed to **init()** method, define them with annotation:
      - public void init(@Param("$target") String target) { … }

yaPOSH
Actions II

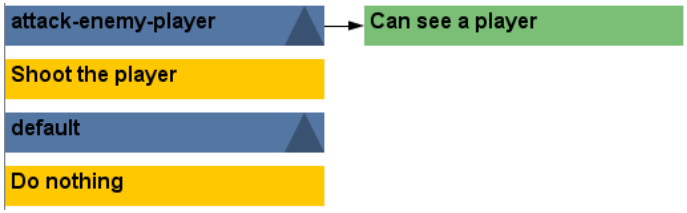


- Actions are expected to return their "state" in the **run()** method – to notify POSH
  - **RUNNING_ONCE**
    - Actions says, I want to be executed for one logic iteration and then I am done
  - **RUNNING**
    - Action says, I am still running and I want to run in next cycles as well
  - **FINISHED**
    - Action says, I am FINISHED and DONE.
    - **This triggers POSH to "replan" – to search for new action – IMMEDIATELLY!** (without waiting for the next update from the environment) See next slides for caveats!
  - **FAILED**
    - An action execution has failed

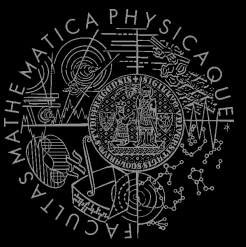
yaPOSH
Actions III – POSH action selection



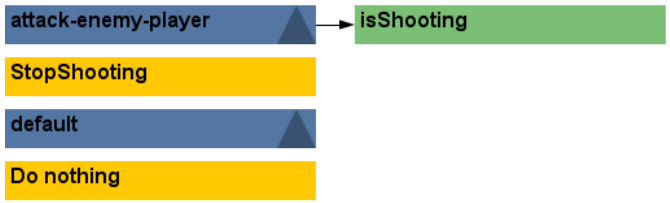
- POSH searches for next action to execute as long as it finds one
- This means if your plan doesn't return any action (no sense matches), the POSH will re-evaluate immediately!
  - **And will be stuck in infinite loop!**
- **Your POSH plan should ALWAYS return action!**
  - **The best way is to have default sense with action doNothing at the bottom of the plan!**
    - **E.g.:**



yaPOSH
Actions IV – FINISHED



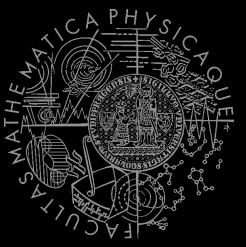
- Action returning in **run()** method **FINISHED** tells POSH to re-evaluate plan immediately to search for a new action
- This can be used to your advantage (parallel actions), but has a caveat!
- Consider plan, where **StopShooting** returns **FINISHED** in **run()** immediately:



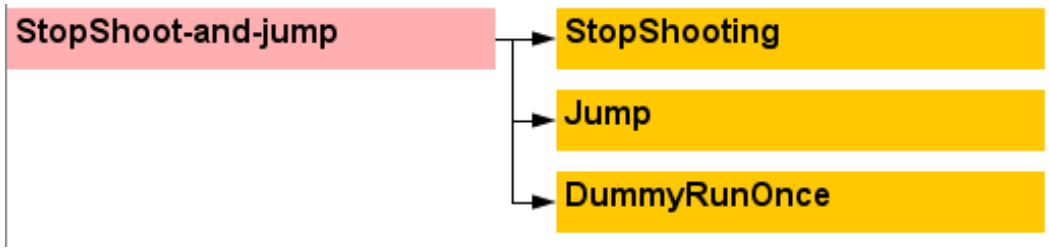
  - Makes sense, because as we send stop shoot command in **init()**, the action is done…
  - The problem is that the POSH re-evaluates the plan immediately to search for a new action and guess what it finds? **StopShooting** again. Why?
    - Because isShooting sense will be returning the same value it was before! The environmental state is not changed. The POSH re-evaluates immediately! We are stuck in infinite loop and no more environmental updates will ever come (even at first glance no exceptions raised).
- For these types of actions always return **RUNNING_ONCE** !

yaPOSH

Actions V – Parallel Actions

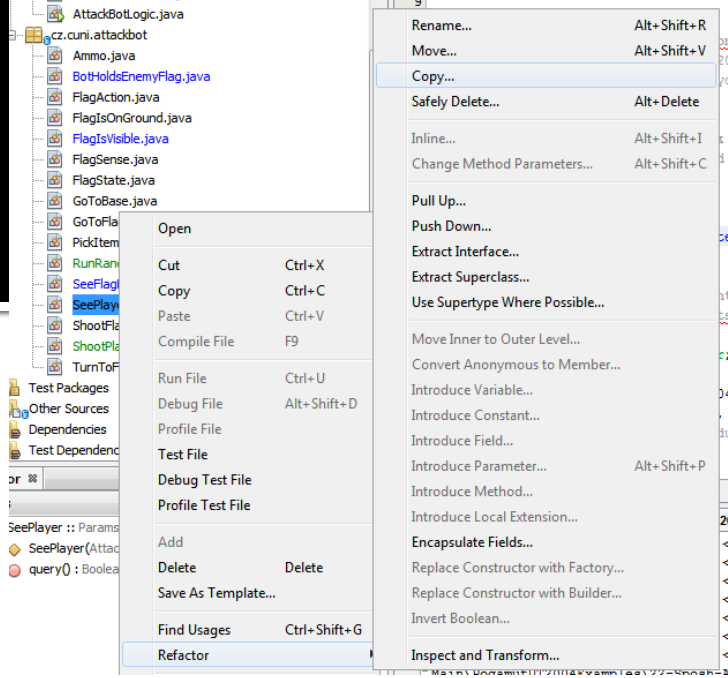


- Now returning **FINISHED** in **run()** method can be used to your advantage if you are careful enough.
- Lets say you want to execute two actions in parallel.
  - Create an action-pattern (drag and drop from action pattern tab)
  - Add the actions you want to execute in parallel (in our example StopShooting and Jump). They will be returning **FINISHED** in **run()** to trigger immediate POSH re-evaluation
  - Add one more dummy action to the end of action pattern returning **RUNNING_ONCE** (to stop POSH plan re-evaluation – otherwise we would be stuck in infinite loop again)
  - Done! You have now action pattern executing two actions in parallel!



yaPOSH

Actions VI – New Action



- How to make a new action?
  - There are no templates yet…
- In NetBeans:
  - Right click on some existing action,
  - Right click the Java class and select refactor and Copy it with a new name
  - Change the action description and human readable name in the annotation before the class declaration

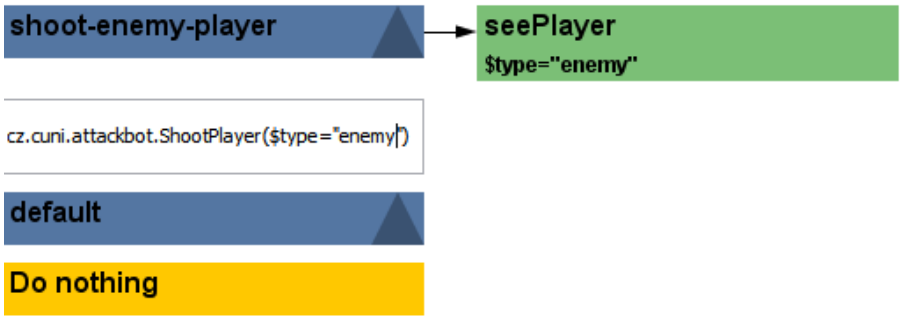
```
@PrimitiveInfo(name="Shoot the player", description="Shoot the player.")
public class ShootPlayer extends ParamsAction<AttackBotContext> {
```

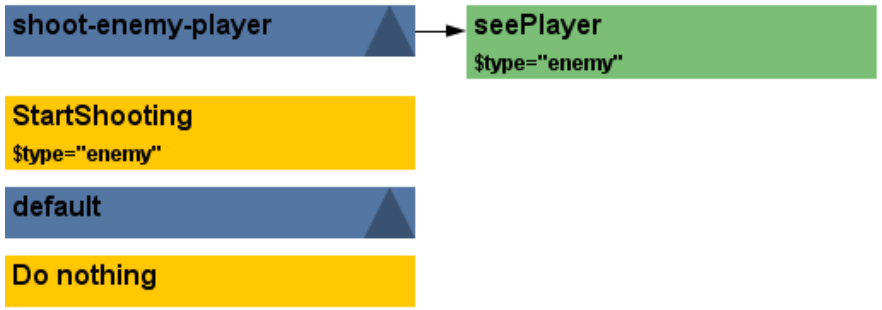
  - In POSH editor click Refresh button in the Senses editor

yaPOSH

Actions VII – Parameterized Action Example

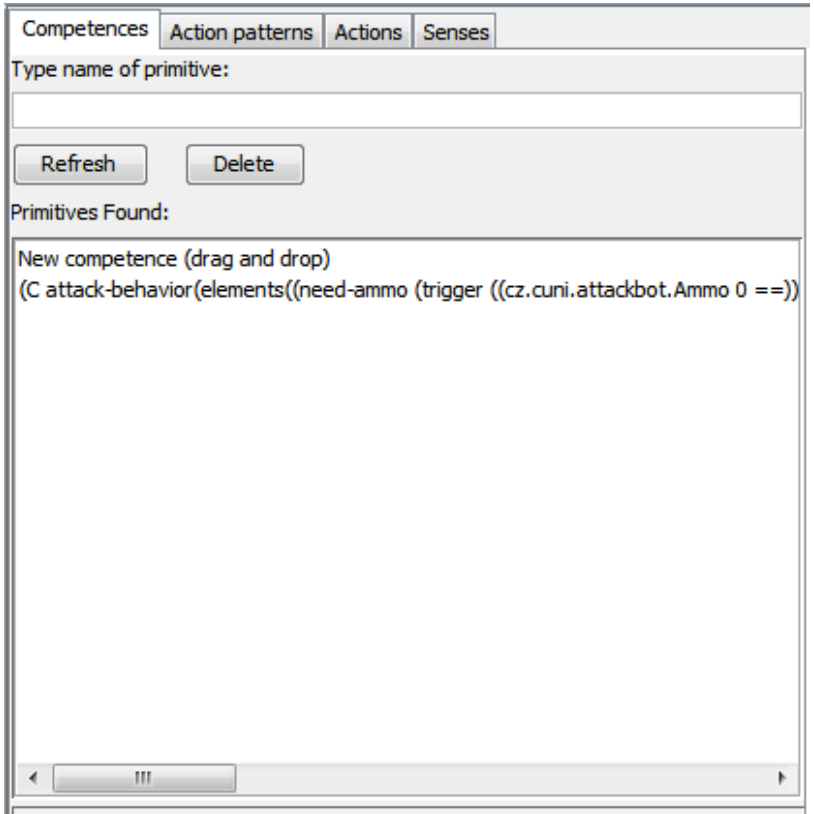

```
@PrimitiveInfo(name="StartShooting", description="Shoot the player.")
public class ShootPlayer extends ParamsAction<AttackBotContext> {

    public ShootPlayer(AttackBotContext ctx) {
        super(ctx);
    }

    public void init(@Param("$type") String type) {
        if (type.contentEquals("friend"))
            ctx.getShoot().shoot(ctx.getPlayers().getNearestVisibleFriend());
        else
            ctx.getShoot().shoot(ctx.getPlayers().getNearestVisibleEnemy());
    }

    public ActionResult run() {
        return ActionResult.RUNNING_ONCE;
    }

    public void done() {

    }
}
```





yaPOSH

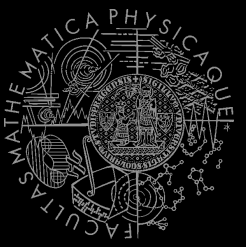
New Action Pattern, New competence



- Are created by drag and dropping from POSH editor from the tabs at the right side of IDE

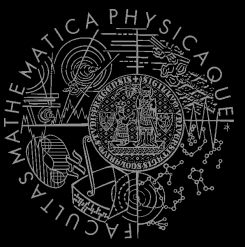
yaPOSH Context

How to access Pogamut modules?



- Every POSH action and sense has context (this.ctx) that contains all Pogamut modules.
- Context is an editable class that is a part of your POSH bot sources, e.g. **AttackBotContext**
- You may use context to store some variables, e.g. Item you are going for or player you are going to fight

yaPOSH
Parameters

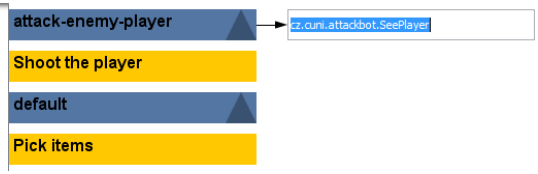


- Competences, action patterns, actions and senses can be parameterized

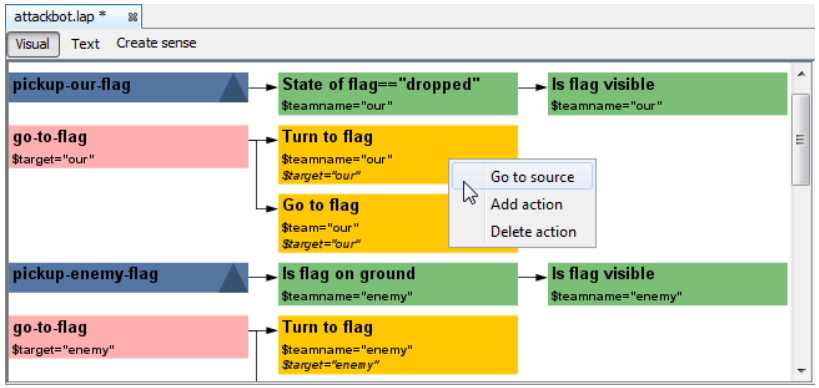
```
(
 (AP go-to-flag
  vars($target="enemy")
   (bot.TurnToFlag($teamname=$target)
    bot.GoToFlag($team=$target)
   )
 )

 (DC life
  (drives
   (
    (pickup-our-flag
     (trigger
      (
       (bot.FlagState($teamname="our")
                      "dropped")
       (bot.FlagIsVisible($teamname="our"))
     ))
     go-to-flag($target="our")
    )
   )
  )
 )
)
```

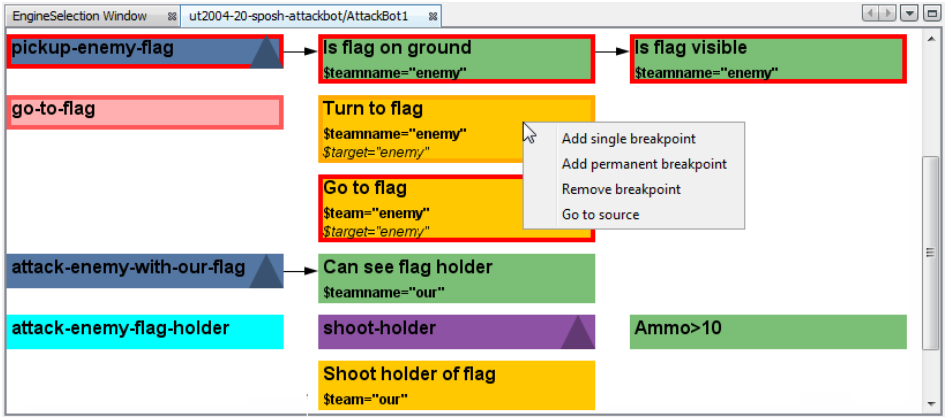
```
@PrimitiveInfo(name        = "Is flag visible",
               description = "our / enemy")
public class FlagVisible
       extends FlagSense<AttackBotContext,Boolean>
{
    public Boolean query(
           @Param("$teamname") String teamname
    ) {
        FlagInfo flag = getFlagInfo(teamname);
        return flag.isVisible();
    }

@PrimitiveInfo(name        = "Turn to flag",
               description = "our / enemy")
public class TurnToFlag
       extends FlagAction<AttackBotContext> {

    public ActionResult run(
           @Param("$teamname") String teamName
    ) {
        FlagInfo flag = getFlagInfo(teamName);
        ctx.getMove().turnTo(flag.getLocation());
        return ActionResult.RUNNING_ONCE;
    }
```

yaPOSH
POSH Editor



- Enables drag and drop
  - Select action or sense you want to add or change from the editor and drag and drop it at desired place
- Double clicking POSH graphical element open editor, right clicking opens element menu
- Support “Go to source”, breakpoints and debugging
- Breakpoints **PAUSE** the **bot <u>AND</u>** the **environment**



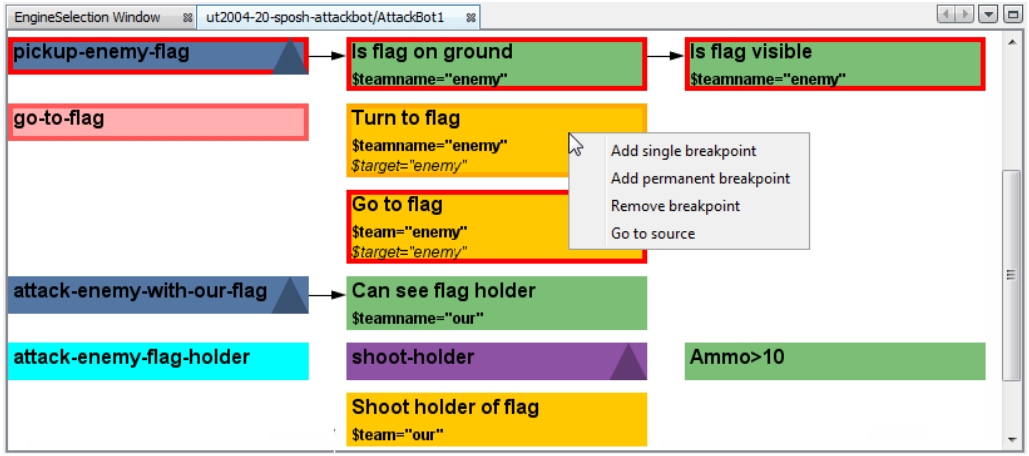


yaPOSH

How to run POSH plan debugger

- Run the bot in **Debug mode** (right click the project, select **Debug**)
- In the Debug toolbar, click the green circle button to enable POSH plan debugger

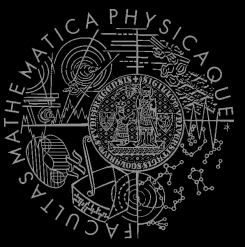
- A window with Debugger appears:



Today's menu

1. Big Picture
2. BOD (Behavior Oriented Design)
3. Gentle POSH introduction
4. **Weapons & Shooting**
   - http://planetunreal.gamespy.com/View.php?view=UT2004GameInfo.Detail&id=26
5. DeathMatch Bot

Weapons & Shooting
Weaponry class



- `this.weaponry`
  - All you wanted to know about UT2004 weapons but were afraid to ask
  - Note that it contains also some obsolete and to-be-deprecated methods…

```
weaponry.getCurrentWeapon()
weaponry.hasWeapon(ItemType)
weaponry.hasLoadedWeapon()
weaponry.hasPrimaryLoadedWeapon()
weaponry.hasSecondaryLoadedWeapon()
weaponry.getLoadedWeapons()
weaponry.changeWeapon()
…
```

Weapons & Shooting
WeaponPreferences

- Weapons' effectiveness depends on distance to target
- Thus you should create different priority list for various "ranges"
- Wrapped in class `weaponPrefs`

```
weaponPrefs.addGeneralPref(ItemType.MINIGUN,  true);
weaponPrefs.addGeneralPref(ItemType.LINK_GUN, false);
```

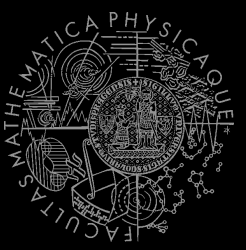
- true -> primary firing mode
- false -> secondary firing mode

```
weaponPrefs.newPrefsRange(CLOSE_COMBAT_RANGE = 300)
   .add(ItemType.FLAK_CANNON, true)
   .add(ItemType.LINK_GUN,    true); // 0-to-CLOSE
weaponPrefs.newPrefsRange(MEDIUM_COMBAT_RANGE = 1000)
   .add(ItemType.MINIGUN,             true)
   .add(ItemType.ROCKET_LAUNCHER,     true); // CLOSE-to-MEDIUM
```

- If range prefs fails, general are used
- You have to experiment! *(== behavior parametrization!)*

More at: **http://pogamut.cuni.cz/pogamut_files/latest/doc/tutorials/10-HunterBot.html**

Weapons & Shooting
Shooting

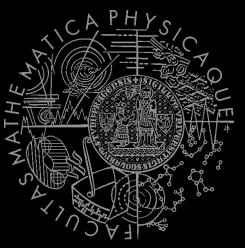


- Shooting with **`WeaponPrefs`** is easy!

```
Player enemy =
   players.getNearestVisiblePlayer();

shoot.shoot(weaponPrefs, enemy);

shoot.shoot(weaponPrefs, enemy,
                ItemType.ROCKET_LAUNCHER);
// do not use rocket launcher

shoot.setChangeWeaponCooldown(millis);
```

Weapons & Shooting

Time your shooting – Cooldown class



- Sometimes you need to perform the behavior "once in a time" => `Cooldown`

```
Cooldown rocketCD = new Cooldown(2000);
                                // millis

if (rocketCD.isCool()) {
   rocketCD.use();
   shoot.shoot(weaponPrefs, enemy);
} else {
   shoot.shoot(weaponPrefs, enemy,
   ItemType.ROCKET_LAUNCHER);
}
```

Weapons & Shooting

Time your behaviors – Heatup class

- Sometimes you need to pursue some behavior for a while => `Heatup`

```
Heatup pursueEnemy = new Heatup(3000);
                                  // millis

if (players.canSeeEnemy()) {
    pursueEnemy.heat();
    // fight the enemy
} else
if (pursueEnemy.isHot()) {
    // pursue the enemy
} else {
    // collect items
}
```

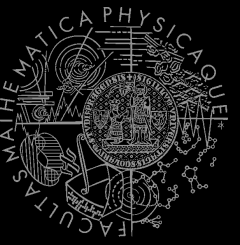
Practice Lesson

Outline

1. Big Picture
2. BOD (Behavior Oriented Design)
3. Gentle POSH introduction
4. Weapons & Shooting
5. **DeathMatch Bot**

Deathmatch Bot

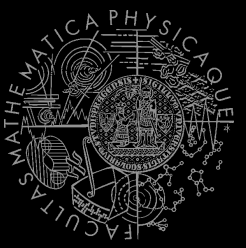
Basics



- Its all about movement on the map
  - Picking the right place to be at
  - Picking the right item to go for
- Knowing when it is worth to change the behavior
  - I am almost at the rocket launcher, but I see enemy player. Will I go for the weapon or start fighting with the player?

Deathmatch Bot

Combat



- Using proper weapon in proper situations
  - `this.weaponPrefs …`
- Knowing how to move in combat
  - Strafing, dodging, jumping
  - Maintaining distance according bot current weapon
  - Facing one direction and move elsewhere (navigation.setFocus(…))
- Beware that jumping and dodging reduces bot accuracy!

Assignment 8
(or Homework)

- Create **DeathMatchBot** in POSH
  - That arms himself and is able to fight an opponent
  - Does not stuck (for long).

Assignment
Cheatsheet

- Access Pogamut modules from POSH actions and senses!
  - `this.ctx.getItems().getSpawnedItems(ItemType.Category.WEAPON)`
  - `MyCollections.getFiltered(Collection, new IFilter<Item>() {…})`
- Handling unreachable items:
  - `this.ctx.getNavigation().addStrongNavigationListener(…STUCK_EVENT…)`
  - `myTabooSet.add() & myTabooSet.filter(…)`
- Specifying weapon preferences:
  - `this.ctx.getWeaponPrefs().addGeneralPref(ItemType.FLAK_CANNON,true)`
    `.addGeneralPref(ItemType.ROCKET_LAUNCHER,true);`

Questions?

I sense a soul in search of answers…

- We do not own the patent of perfection (yet…)
- In case of doubts about the assignment, tournament or hard problems, bugs don't hesitate to contact us!
  - Jakub Gemrot (Monday practice lessons)
    - jakub.gemrot@gmail.com
  - Michal Bída (Thursday practice lessons)
    - michal.bida@gmail.com