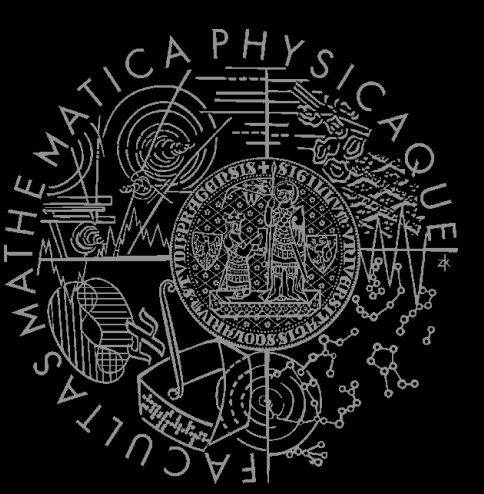
University of West Bohemia, Pilsen
2nd March 2015



UT2004 bots made easy!

Pogamut 3

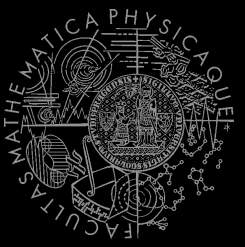
Navigation & Items, Weapons & Shooting, CTF

Practice Lesson
Outline

Pogamut 3 Platform

1. **Pogamut World Abstraction**
2. Navigation
3. Items & Weapons & Shooting
4. Capture the Flag (CTF)

Pogamut World Abstraction
Basics



Objects (IWorldObject):

- Player
- Item
- NavPoint
- Self
- IncomingProjectile

Events (IWorldEvent):

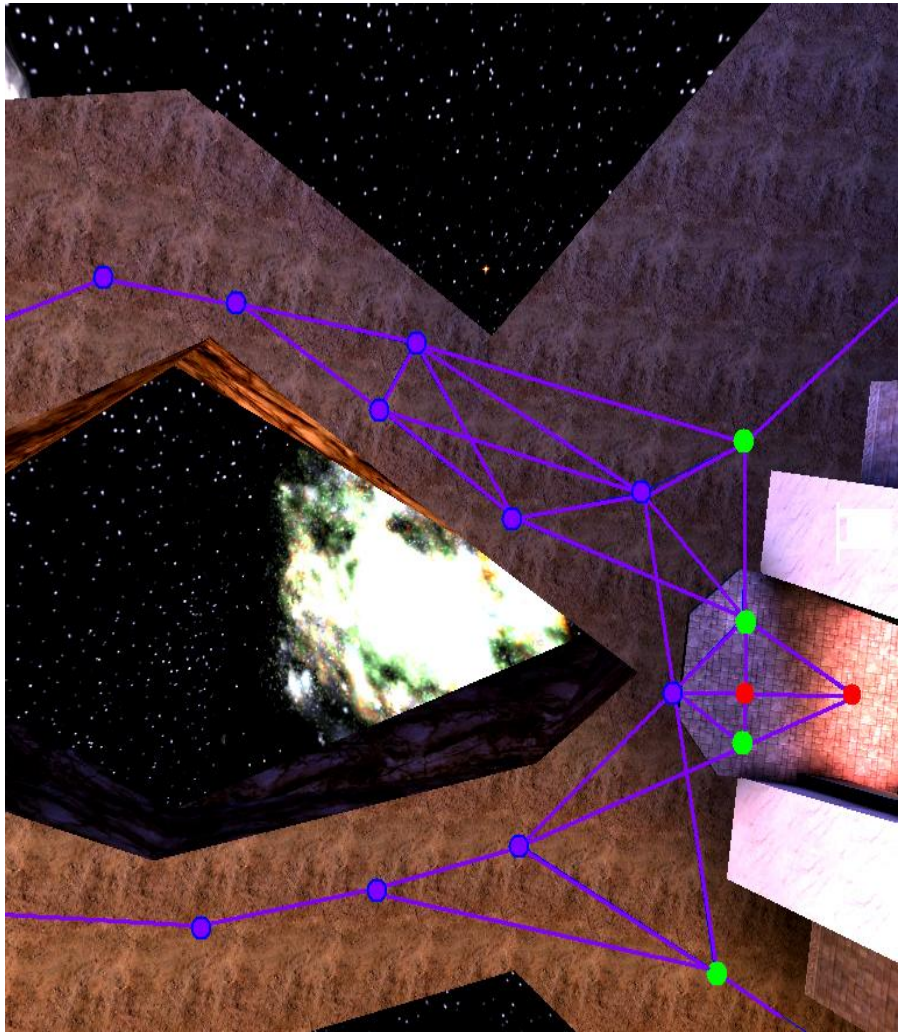
- HearNoise & HearPickup
- BotDamaged & BotKilled
- PlayerDamaged & PlayerKilled,
- Bumped
- GlobalChat

- Use modules, listeners and Pogamut helper classes!
  - this.players, this.items, this.info …
  - MyCollections, DistanceUtils

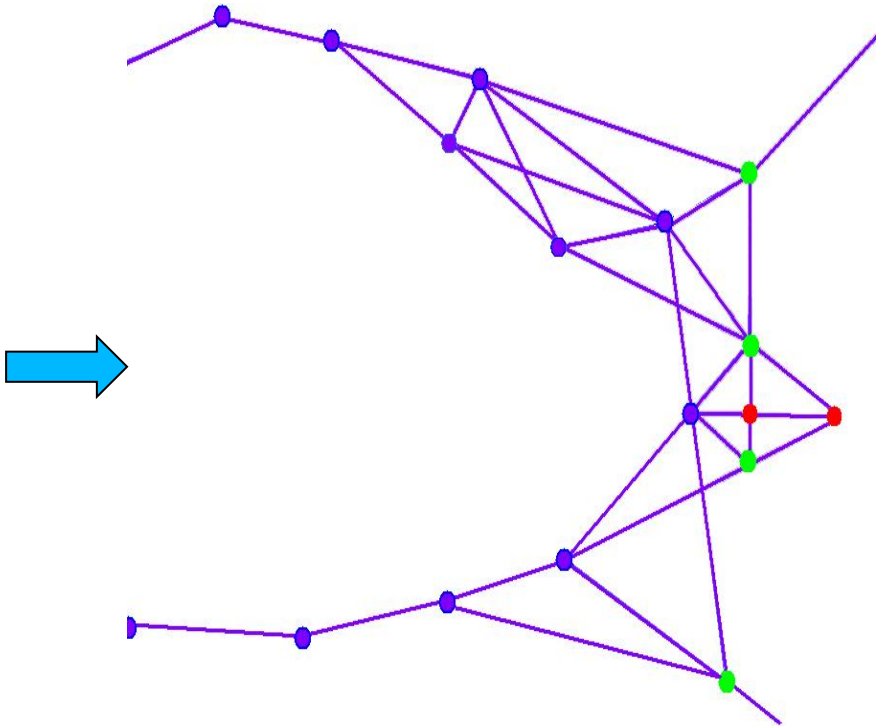
```
if (this.players.canSeePlayers()) {  …  }

@EventListener(eventClass = GlobalChat.class)
public void chat(GlobalChat chatEvent) {
    …
}
```

UT2004 World Abstraction

Navigation graph

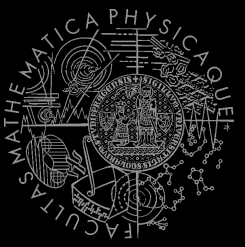


#Navpoints in the map = 100 – 5000



UT2004 World Abstraction

Underlaying classes – low level API



Classes of interest:

NavPoint, NavPointNeighbourLink, Item

ILocated, Location, DistanceUtils

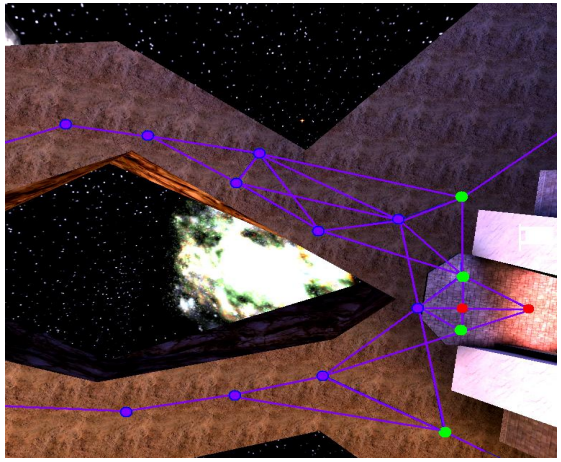
ItemType, ItemType.Category

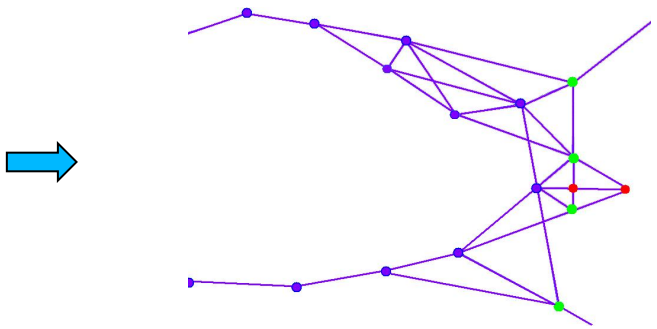
ItemDescriptor

Methods of interest:

```
this.items.getAllItems(ItemType)
this.descriptors.getDescriptor(ItemType)
this.world.getAll(NavPoint.class)
this.world.getAll(Item.class)
NavPoint.getOutgoingEdges()
NavPoint.getIncomingEdges()
```

!!!





UT2004 World Abstraction

Nav link/NavPoint types

- **NavPoints**
  - JumpPad
  - Lift
  - Teleport
  - Door
  - PlayerStart
  - SnipingSpot
  - InventorySpot
  - …
- **Nav links**
  - Walk
  - Jump
  - Lift
  - Door
  - DoubleJump
  - …

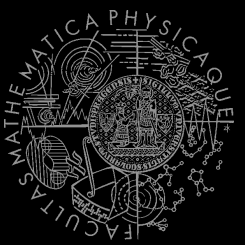
Practice Lesson
Outline

Pogamut 3 Platform

1. Pogamut World Abstraction
2. **Navigation**
3. Items & Weapons & Shooting
4. Capture the Flag (CTF)

Navigation

Step by step

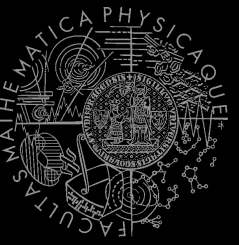


1. Decide where to go
2. Plan the path (list of navpoints)
3. Follow the path
   - Handle jumps&lifts along the way!
     - Do you know right constants?
   - World is non-deterministic, be sure to check how the action is executing!
     => `IStuckDetector` implementations

Don’t worry it’s already wrapped up ☺

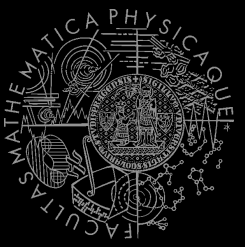
Navigation

Stages



1. Decide where to go (Decision making!)
   - `items.getSpawnedItems(ItemType)`
   - `DistanceUtils.getNearest(…)`
   - `MyCollections.getRandom(…)`
   - `fwMap.getNearest(…)`
2. Plan and follow the path
   - UT2004Navigation (this.navigation)

Navigation
UT2004Navigation

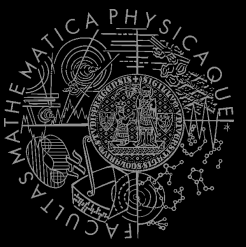


- Complete navigation wrapper
  - **`UT2004Navigation`**`(…, UT2004PathExecutor, FloydWarshallMap, …) (this.navigation)`
  - Handles both path planning & path following
  - Can be called repeatedly
- Main methods
  - `navigation.navigate(…)`
  - `navigation.isNavigating()`
- Uses
  - `FloydWarshallMap (this.fwMap)`
  - `StuckDetectors`
  - `UT2004PathExecutor`

Navigation
FloydWarshallMap

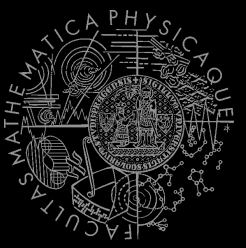
- Pogamut path planner using **Floyd Warshall** algorithm ($O(n^3)$!)
  - Used by UT2004Navigation
  - Access by `this.fwMap`
- Methods of interest
  - `fwMap.getNearest…(…)`
    - Works the same as in DistanceUtils, except the distance is measured by the path length
    - Its ok to "spam" it (e.g. checking all items in each step), the nowadays computers can handle it

Navigation
Modifying the navigation graph



- NavigationGraphBuilder
  - Access by `this.navBuilder`
- Methods of interest
  - `navBuilder.removeEdgesBetween(…)`
- If you use `navBuilder` in **botInitalized** method, everything will be applied automatically
  - Otherwise, call `fwMap.refreshPathMatrix()`
    - $O(n^3)$!!

Navigation
StuckDetectors



- Navigation Uses three stuck detectors
- **UT2004TimeStuckDetector(bot, 3000)**
  - if the bot does not move for 3 seconds consider it is stuck (check small velocity delta)
- **UT2004PositionStuckDetector()**
  - watch over the position history of the bot, if the bot does not move sufficiently enough, consider that it is stuck
  - DEFAULT_HISTORY_LENGTH, DEFAULT_MIN_DIAMETER, DEFAULT_MIN_Z
- **UT2004DistanceStuckDetector()**
  - counts how many times the bot was getting closer to the target and how many times it was getting farther (if it oscillates more than two times -> STUCK)

Navigation
Listening for navigation events

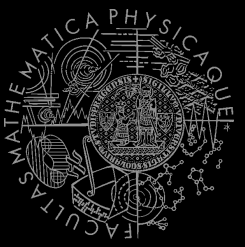
- With a FlagListener! Add one with method addStrongNavigationListener

```
this.navigation.addStrongNavigationListener(
                    new FlagListener<NavigationState>() {
                        @Override
                        public void flagChanged(NavigationState
    changedValue){
                            switch (changedValue) {
                                case STUCK:
                                break;
                                case STOPPED:
                                break;
                                case TARGET_REACHED:
                                break;
                                case PATH_COMPUTATION_FAILED:
                                break;
                                case NAVIGATING:
                                break;
                            }
                        }
        });
```

Navigation
Path following hell

- **UT2004PathExecutor**
- Custom Pogamut path following code
  - Heavily tweaked for UT2004 and game update frequency 4 Hz (250 ms per synchronous batch)
- The good
  - Works decently on non-complex maps
  - You don't have to do it yourself
- The bad
  - Has problems handling complex links
  - Spaghetti code

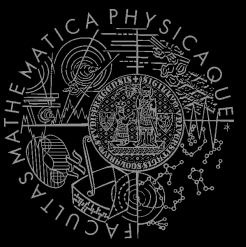
Navigation
UT2004AStar



- When Floyd Warshall is not enough…
- **UT2004AStar** (access by this.aStar)
  - this.aStar.findPath(from, to, IPFMapView);
- Implement your own custom **IPFMapView:**

```
new IPFMapView<NavPoint>() {

        public Collection<NavPoint> getExtraNeighbors(NavPoint node,
    Collection<NavPoint> mapNeighbors) {}

        public int getNodeExtraCost(NavPoint node, int mapCost) {}

        public int getArcExtraCost(NavPoint nodeFrom, NavPoint nodeTo, int mapCost) {}

        public boolean isNodeOpened(NavPoint node) {}

        public boolean isArcOpened(NavPoint nodeFrom, NavPoint nodeTo) {}
}
```

Assignment 1



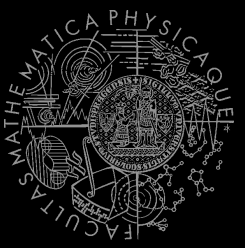
- Let's create **NavigationBot**
  - Choose NavPoint at random
  - Run to that NavPoint
  - Iterate
- How to detect that the bot has stuck?
- What if the location is currently unreachable?
  - See `TabooSet` class

Practice Lesson
Outline

Pogamut 3 Platform

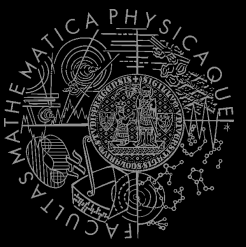
1. Pogamut World Abstraction
2. Navigation
3. **Items & Weapons & Shooting**
4. Capture the Flag (CTF)

Items, Weapons, Shooting
Items – basics



- `Item (module this.items !)`
  - More "spawning location" than item
  - Unique `UnrealId` => Can be used in `Set`, `Map`
  - `ILocated` ~ `getLocation()` ~ X, Y, Z
  - `IViewable` ~ `isVisible()`
  - Always has corresponding `NavPoint` instance
    - `NavPoint itemNP = item.getNavPoint()`
  - Described by `ItemType`
    - `item.getType()`

Items, Weapons, Shooting
ItemType



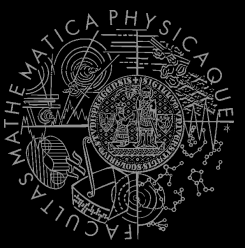
- `ItemType`
  - Enum holding concrete type of the item
  - Part of some `ItemType.Category`
    - Categories are divided based on what items are intended to do
    - `ItemType.Category.HEALTH`
    - `ItemType.Category.ARMOR`
    - `ItemType.Category.SHIELD`
    - `ItemType.Category.WEAPON`
    - `ItemType.Category.AMMO`

Items, Weapons, Shooting
Items

- Agent module: `items`
  - `.getAllItems()`
  - `.getVisibleItems(ItemType/)`
  - `.getSpawnedItems(ItemType)`
  - `.isPickable(Item)`
- `DistanceUtils`
  - `.getNearest(Collection<Ilocated>)`
  - `.getNthNearest(n,Collection<Ilocated>)`
- `fwMap`
  - `.getNearestItem(Collection<Item>)`

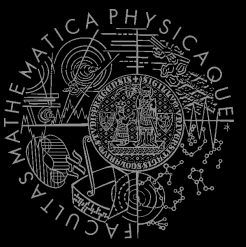
Items, Weapons, Shooting

Always collect interesting items



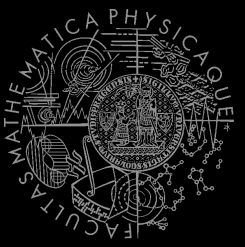

```
ItemType.FLAK_CANNON
        .MINIGUN
        .LIGHTING_GUN
        .ROCKET_LAUNCHER
        .LINK_GUN

ItemType.SUPER_HEALTH
        .SUPER_ARMOR
        .SHIELD_PACK
        .SUPER_SHIELD_PACK
        .U_DAMAGE_PACK
```

Assignment 2



- Alter **NavigationBot** into **CollectorBot**
  - Collect interesting items
  - How to check that your bot can pick some item?
    - `items.isPickable(Item)`
  - How to be sure that your bot has picked the item up?
    - `ItemPickedUp.class` event `@EventListener(eventClass= ItemPickedUp.class)`
    - Be sure to handle "thin" items!
  - How to avoid unreachable items?
    - Use `TabooSet`

Items, Weapons, Shooting
ItemDescriptor(s)



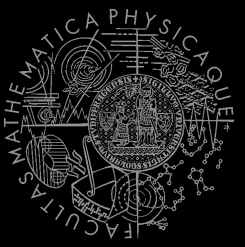
- Every item is “well” described

```
Item item =
    items.getAll(ItemType.Category.WEAPONS).values()
    .iterator().next();
WeaponDescriptor weaponDesc =
    (WeaponDescriptor)
    descriptors.getDescriptor(item.getType());
if (weaponDesc.getPriDamage() > 50) {
…
}
```

- Ammo/Armor/HealthDescriptor available as well

Items, Weapons, Shooting

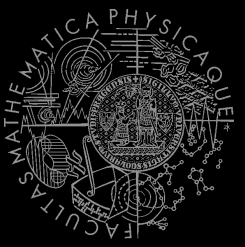
UT2004 weapons guide I – the weak



- `ItemType.SHIELD_GUN (DEFAULT)`
  - Melee weapon (can be charged)
  - Secondary mode – shield
- `ItemType.ASSAULT_RIFLE (DEFAULT)`
  - Weak, basic, inaccurate (but can be double wielded)
  - Secondary mode – grenades (charged)
- `ItemType.BIO_RIFLE`
  - Fires green blobs, short range, defense weapon
  - Secondary mode – charged (big blob)
- `ItemType.LINK_GUN`
  - Primary fires rather slow, but decent projectiles
  - Secondary – medium-to-short range beam

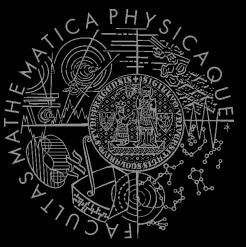
Items, Weapons, Shooting

UT2004 weapons guide II – the strong



- `ItemType.FLAK_CANNON`
  - Shotgun style weapon – deadly at short range
  - Sec. mode is a grenade launcher
- `ItemType.MINIGUN`
  - Choose between rapid fire but less accuracy (pri. mode) or slower fire and more accuracy (sec. mode)
- `ItemType.SHOCK_RIFLE`
  - Pri. mode is very accurate with medium damage
  - Sec. mode fires slow moving projectiles, that can be detonated by pri. fire making a big explosion (tricky to do though)
- `ItemType.LIGHTING_GUN / SNIPER_RIFLE`
  - Sniper rifle – precise, can one-shot others by a headshot
  - Bots can use only pri. fire (sec. is zoom)

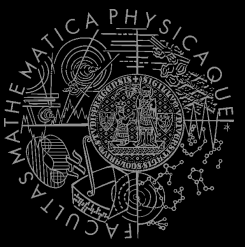
Items, Weapons, Shooting
UT2004 weapons guide III – mayhem



- `ItemType.ROCKET_LAUNCHER`
  - Good old rocket launcher, rockets have splash damage (beware!)
  - Secondary mode can charge up to three rockets
- `ItemType.REDEEMER`
  - Unleash nuclear mayhem! (big splash damage radius)
  - Bots can use only primary firing mode!

- `ItemType.U_DAMAGE_PACK`
  - Not enough damage? Grab DOUBLE DAMAGE pack and double your damage output!

Items, Weapons, Shooting
Weaponry class



- `this.weaponry`
  - all you wanted to know about UT2004 weapons but were afraid to ask

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

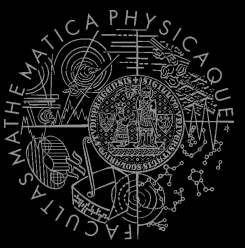
Items, Weapons, Shooting
WeaponPreferences

- 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);

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
```

- true -> primary firing mode
- false -> secondary firing mode
- If range prefs fails, general are used
- You have to experiment! *(== behavior parametrization!)*

Items, Weapons, Shooting
Shooting



- Shooting with `WeaponPrefs` is easy!

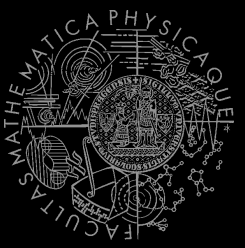
```
Player enemy =
    players.getNearestVisiblePlayer();

shoot.shoot(weaponPrefs, enemy);

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

shoot.shoot(weaponPrefs, enemy);

shoot.setChangeWeaponCooldown(millis);
```

Items, 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);
}
```

Items, 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
}
```

Assignment 3

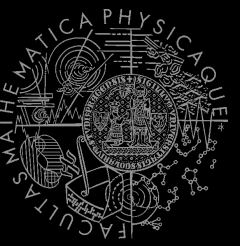
- Alter **CollectorBot** into **HunterBot**
  - Prefer weapons when collecting items
  - Implement shooting behavior
  - Configure & Use `weaponPrefs`
  - Try to run directly towards your opponent
  - Create hunting behavior

Practice Lesson
Outline

Pogamut 3 Platform

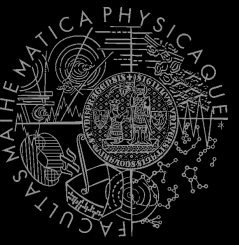
1. Pogamut World Abstraction
2. Navigation
3. Items & Weapons & Shooting
4. **Capture the Flag (CTF)**

Capture the Flag (CTF)
Rules



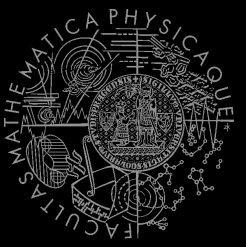
- Players/bots are divided into two teams (red and blue).
- Each team has a flag in his base.
- The goal of the team is to capture the flag of the opposite team and bring it to their home base.
- When managed, the team scores 1 point.
  - Team can only bring opposite flag home and score a point, if the team flag is in team home base!
- If the flag is dropped it will be returned to home base after some time.

Pogamut CTF support
Bases & game status



- **CTF module**
  - this.ctf
- **Where are the bases?**
  - this.ctf.getOurBase();
  - this.ctf.getEnemyBase();
- **Whats the game status?**
  - this.ctf.canOurTeamScore();
  - this.ctf.canEnemyTeamScore();
- **Am I winning?**
  - game.getTeamScores();
  - info.getTeamScore();

Pogamut CTF support II
Flags



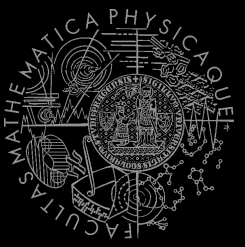
- **I want my flag!**
  - Flag is represented by **FlagInfo** object.
  - this.ctf.getOurFlag();
  - this.ctf.getEnemyFlag();
- **Is someone messing with my flag?**
  - this.ctf.isOurFlagHome();
  - this.ctf.isOurFlagHeld();
- **How about enemy flag?**
  - this.ctf.isEnemyFlagHome();
  - this.ctf.isEnemyFlagHeld();

Pogamut CTF support III
Team communication

- Use **SendMessage** command.
  - this.act.act(new SendMessage().setTeamIndex(info.getTeam()).setText("Help"));
- Listen to team message with **TeamChat** event.

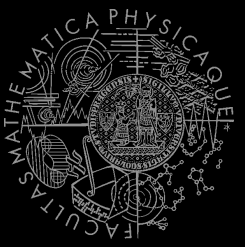
```
@EventListener(eventClass = TeamChat.class)
public void teamChat(TeamChat event) {
        …
}
```

Assignment 4



- Alter **HunterBot** into **CTFBot**
  - Arm yourself before going into action!
  - Try to get enemy flag!
  - Try to get your flag, if it is stolen!

Assignment 5 (bonus)



- Create **RocketDodgeBot** dodging enemy rockets!
  - Dodge commands works properly in 3.5.1-SNAPSHOT

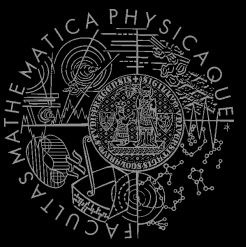
Navigation – detailed

Path planner & Path executor

1. Plan the path (list of navpoints)
   - `pathPlanner.computePath( ILocated from, to)`
     - Watch out for UT2004 quirks! Max 31 navpoints per path (+ starting position location == 32 path points).
   - `fwMap.computePath(NavPoint from, to)`
     - Plans path only between `NavPoints`
2. Follow the path
   - `pathExecutor.followPath(path)`
   - `pathExecutor.isExecuting()`
   - Watch out for its statefullness!

Navigation – detailed

fwMap vs. pathPlanner



`pathPlanner`

- Path is planned at UT2004 => slower
- Graph is fixed
- May plan everywhere
- Has limit ~ 32 path points

`fwMap`

- Floyd-Warshall
  - O(n) path retrievel
- Graph may be altered
- Can't plan to all locations

`pathExecutor` works with both!