findClosestSafeSpot looks at the stored list of area threat locations (e.g. grenades, burning vehicles), and finds the closest safe spot.

```
broadcastEvent(new ev_Move(findClosestSafeSpot()));
```

- `ev_GrenadeNearby/storeThreatLocation(grenade)`
- `ev_GrenadeExploded/removeThreatLocation(grenade)`
- `ev_VehicleDestroyed/removeThreatLocation(vehicle)`
- `ev_BurningVehicleNearby/storeThreatLocation(vehicle)`
- `ev_EvacuateArea`
determineFleePoint will get characterMemorizer.VisibleEnemies and characterMemorizer.EnemiesOutOfSight, then use them to determine progressively more distant safe spots to flee to.
determineFleeTarget takes in a list of nearby enemies and returns a position that is the chosen spot to which to flee.
MeleeCombatExecutor

Being "in melee range" means the enemy is close but not necessarily in striking distance.
RangedCombatExecutor

Diagram of state transitions and event handling:

- **Fighting**
  - `no_grenades`
  - `has_grenades`
  - `no_fighting`

- **Targeting**
  - `selectRangedTarget(characterMemory.getVisibleEnemies())`

- **Firing**
  - `broadcastEvent(new ev_FireRangedWeapon())`
  - `lineOfFireUnobstructed()`
  - `needs_reload`
  - `broadcastEvent(new ev_ReloadWeapon())`
  - `ev_ClipEmpty`

- **State 5**
  - `broadcastEvent(new ev_Move(repositionLocation(target)))`
  - `lineOfFireUnobstructed()`

- **Throwing**
  - `startGrenadeTimer()`

- **Event Handling**
  - `ev_RangedCombat`
  - `ev_StopCombat`
  - `ev_HasGrenades`
  - `ev_NoGrenades`

- `ev_GrenadeTimerUp` is generated by the Start grenade timer method.

- `inGrenadeRange(target)`
  - `broadcastEvent(new ev_ThrowGrenade())`
inactive

\[ ev\_StartExecuteSearch \]

\[ ev\_StopExecuteSearch / broadcastEvent(ev\_StopMove()) \]

searching

getting_list

\[ \text{characterMemorizer.getEnemiesOutOfSight...} \]

\[ ev\_MoveComplete \]

\[ ev\_MoveFailed \]

looking

\[ [\text{enemiesOutOfSight.size}>0] / broadcastEvent(new ev\_Move(searchTarget())) \]

\[ [\text{enemiesOutOfSight.size}==0] / broadcastEvent(new ev\_Wander()); \]

wandering

\[ broadcastEvent(new ev\_StopWander()) \]
Waiting

- ev_StopTakeCover

Covered

- ev_StopTakeCover
- ev_AttackAbated

Taking Cover

- ev_TakeCover
- ev_MoveComplete
- ev_UnderAttack
- broadcastEvent(new ev_Move(determineCoverPoint()));
- broadcastEvent(new ev_StopMove());
UseItemExecutor

```
no_item

has_item

using_item

broadcastEvent(new ev_ActuateUseItem())
```

- `ev_ItemAcquired`
- `ev_UseItem`
- `ev_ItemUseFailed`
- `ev_ItemUsed`
This Statechart uses 4 methods to effect vehicle combat. First, the `selectTarget(Players[])` selects the most appropriate attack target. Then, the `isTargetInFront` method quickly determines if the target is in a cone in front of the NPC. If not, the `attackRunStartSpot` method will select an appropriate spot from which to attack. Finally, the `calculateAttackRunTarget` will choose a spot behind the target, in an attempt to run down the target.
not_wandering

wandering

broadcastEvent(new ev_Move(wanderTarget())); resetTimer()

broadcastEvent(new ev_Move(wanderTarget())); resetTimer()

ev_WanderTimer

ev_WanderTimer is generated internally when the wander timer goes off.