

## Adding reload into the Python API

- reload is partially implemented and these notes will help you complete it
- the files which need to be modified are split into two groups
  - server side (dhewm3 engine)
  - client side (python)
- server side files: `neo/game/Player.cpp`, `neo/game/Player.h`,  
`neo/game/ai/pybot.cpp` `neo/game/ai/pybot.h`
- client side files: `python-bot/botbasic.py`, `python-bot/botcache.py`, `python-bot/botlib.py`, `python-bot/python_doommarine.py`

## python-bot/botlib.py

- add this code above the `sync` method
- this code calls upon the `_cache` library to `reload_weapon`

```
#  
# reload_weapon - reload the current weapon  
#                 It returns the amount of ammo left.  
#  
  
def reload_weapon (self):  
    return self._cache.reload_weapon ()
```

## python-bot/botcache.py

- add this code above the sync method

```
#  
# reload_weapon - reload the current weapon  
#                 It returns the amount of ammo left.  
#  
  
def reload_weapon (self):  
    return self._basic.reload_weapon ()
```

## python-bot/botbasic.py

- rename the method `reloadWeapon` into `reload_weapon` for consistency
- python client side changes are complete
  - with the exception of the test code in `python-bot/python_doommarine.py` which is left as an exercise for the reader

## dhewm3 server side changes for reload\_weapon

neo/game/Player.cpp

```
/*
=====
idPlayer::reload_weapon
=====
*/
int idPlayer::reload_weapon (void) {
    if ( gameLocal.isClient ) {
        return -1;
    }
    if ( spectating || gameLocal.inCinematic || influenceActive ) {
        return -1;
    }
    if ( weapon.GetEntity() && weapon.GetEntity()->IsLinked() ) {
        weapon.GetEntity()->Reload ();
        return inventory.ammo[currentWeapon];
    }
    return -1;
}
```

## dhewm3 server side changes for reload\_weapon

- add the reload\_weapon declaration to the Player class
  - add it under ChangeWeapon

neo/game/Player.h

```
void select (int bitmask);  
int ChangeWeapon (int new_weapon);  
int reload_weapon (void);
```

## dhewm3 server side changes for reload\_weapon

- the server side has partial support for reload weapon but it is currently broken and we will fix it
- fix the code in `rpcReloadWeapon`

## dhewm3 server side changes for reload\_weapon

neo/game/ai/pybot.cpp

```
/*
 * rpcReloadWeapon - return the amount of ammo available for the current weapon
 *                   after reloading.
 */
void pyBotClass::rpcReloadWeapon (void)
{
    char buf[1024];
    int ammo;

    if (protocol_debugging)
        gameLocal.Printf ("rpcReloadWeapon call by python\n");
    if (rpcId > 0)
        ammo = dictionary->reload_weapon (rpcId);
    else
        ammo = 0;
    idStr::snPrintf (buf, sizeof (buf), "%d\n", ammo);
    buffer.pyput (buf);
    state = toWrite;
}
```



## Add reload\_weapon to the dict class

- check that the declaration also exists in the dict class
- now add this method above the health method

```
/*  
 * reload_weapon - reload the current weapon and return the  
 *                  ammo available for the current weapon.  
 */  
  
int dict::reload_weapon (int id)  
{  
    return entry[id]->reload_weapon ();  
}
```

## reload\_weapon in the item class

- change existing declaration to return an `int`. The reload will return the amount of ammo left.
- change the declaration of `reload_weapon` in class `item`

■ `neo/game/ai/pybot.cpp`

```
int health (void);  
int angle (void);  
int reload_weapon (void);  
bool aim (idEntity *enemy);  
int turn (int angle, int angle_vel);  
idEntity *getIdEntity (void);
```

## reload\_weapon in the item class

neo/game/ai/pybot.cpp

```
/*
 * reload_weapon
 */
int item::reload_weapon (void)
{
    switch (kind)
    {
        case item_monster:
            assert (false);
            return 0; // ignore
            break;
        case item_player:
            return idplayer->reload_weapon ();
    }
    assert (false);
    return 0;
}
```