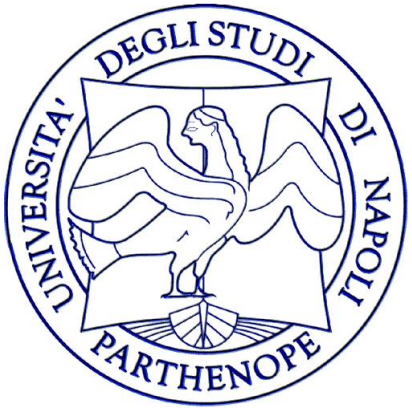
Universitá degli Studi di Napoli "Parthenope"
Scuola interdipartimentale delle Scienze, dell'Ingegneria e della Salute
Dipartimento di Scienze e Tecnologie

Corso di laurea in Informatica



Tesi di laurea triennale

Analisi delle interazioni molecolari e sviluppo di un software per l'automatizzazione del docking molecolare

Preparazione dei ligandi e dei recettori, docking ed estrazione dei legami

Relatore
Ch.mo
Prof. Angelo Ciaramella

Correlatore
Dott. Ferdinando Febbraio

Laureando
Alfredo Mungari
Matr. 0124002134

Anno Accademico 2021-2022

Indice

1 Introduzione **6**

1.1 Docking Molecolare 9
1.2 Simulazione 12
1.3 Software per il docking molecolare 12
1.4 Idea e sviluppo 14
1.5 Contenuto della tesi 15

2 Applicazione realizzata **16**

2.1 Installazione 16
2.2 Modalità di utilizzo 17
2.3 Dati in input 19
2.4 Preparazione dei ligandi e dei recettori 21
2.4.1 Preparazione dei ligandi tramite script 23
2.4.2 Preparazione dei ligandi tramite GUI . 29
2.4.3 Preparazione dei recettori tramite script 32
2.4.4 Preparazione dei reccettori tramite GUI 42
2.5 Docking 45

3 Applicazione realizzata **46**

3.1 Installazione 46
3.2 Modalità di utilizzo 47

3.3 Dati in input 49
3.4 Preparazione dei ligandi e dei recettori 51
3.4.1 Preparazione dei ligandi tramite script 53
3.4.2 Preparazione dei ligandi tramite GUI . 59
3.4.3 Preparazione dei recettori tramite script 62
3.4.4 Preparazione dei reccettori tramite GUI 72
3.5 Docking 75

4 Conclusioni **76**

A Tabella dei ligandi **77**

B Tabella dei recettori **81**

Elenco delle figure

1.1 Esemplare adulto di Apis Mellifera 8
1.2 Rappresentazione dei due ligandi ibuprofene a sinistra e celecoxib a destra che hanno effettuato il docking con un enzima di COX2 . . . 11
1.3 Processo del docking molecolare 14

2.1 Sezione **Home Page** della GUI 19
2.2 Query dei recettori 20
2.3 Proteine dei recettori dell'apis mellifera 21
2.4 Sezione **Preparation** della GUI 22
2.5 Sezione **Ligands** della GUI 30
2.6 **Progress bar** dei file *.sdf* 31
2.7 **Progress bar** dei file *.pdb* 31
2.8 **Progress bar** dei file *.pdbqt* 31
2.9 Messaggio processo completato con successo . 32
2.10 Messaggio di errore 32
2.11 Sezione **Receptors** della GUI 43
2.12 **Progress bar** delle proteine 44
2.13 Messaggio processo completato con successo . 44
2.14 Messaggio di errore 45

3.1 Sezione **Home Page** della GUI 49
3.2 Query dei recettori 50

3.3 Proteine dei recettori dell'apis mellifera 51
3.4 Sezione **Preparation** della GUI 52
3.5 Sezione **Ligands** della GUI 60
3.6 **Progress bar** dei file *.sdf* 61
3.7 **Progress bar** dei file *.pdb* 61
3.8 **Progress bar** dei file *.pdbqt* 61
3.9 Messaggio processo completato con successo . 62
3.10 Messaggio di errore 62
3.11 Sezione **Receptors** della GUI 73
3.12 **Progress bar** delle proteine 74
3.13 Messaggio processo completato con successo . 74
3.14 Messaggio di errore 75

Elenco delle tabelle

2.1 Ligandi non scaricati 27

3.1 Ligandi non scaricati 57

A.1 Tabella dei ligandi in input 80

B.1 Tabella dei recettori in input 81

Capitolo 1

Introduzione

Le api recano importanti benefici e servizi ecologici per la società. Con l'impollinazione le api svolgono una funzione strategica per la conservazione della flora, contribuendo al miglioramento ed al mantenimento della biodiversità.
In botanica, l'impollinazione è quel processo che consiste nel trasporto dei pollini dalla parte maschile e quella femminile dell'apparato riproduttivo delle piante. Grazie ad agenti atmosferici e sopratutto al lavoro incessante degli insetti impollinatori, soprattutto le api, il polline viene trasportato da una pianta all'altra rendendo possibile la fecondazione di un'essenza vegetale della stessa specie e la conseguente produzione di semi e frutti. Una diminuzione delle api può quindi rappresentare una importante minaccia per gli ecosistemi naturali in cui esse vivono. L'agricoltura, d'altro canto, ha un enorme interesse a mantenere le api quali efficaci agenti impollinatori. La Food and Agriculture Organization - FAO ha informato la comunità internazionale dell'allarmante riduzione a livello mondiale di insetti impollinatori, tra cui Apis mellifera, le api da miele. Circa l'84% delle specie di piante e l'80% della

produzione alimentare in Europa dipendono in larga misura dall'impollinazione ad opera delle api ed altri insetti pronubi [1]. Pertanto, il valore economico del servizio di impollinazione offerto dalle api risulta fino a dieci volte maggiore rispetto al valore del miele prodotto.
Da un rapporto dell'Unione Internazionale per la Conservazione della Natura (LUCN) risulta che il 10% delle specie selvatiche di api (apis mellifera) sarebbe in via di estinzione e un altro 5% sarebbe a rischio. Una delle principali cause sono i pesticidi, i quali influenzano l'apprendimento, la capacità riproduttiva, i comportamenti sociali di questi insetti e l'orientamento.
La mortalità delle api (Apis mellifera) è un fenomeno che si acuisce soprattutto in primavera e che rischia di compromettere la fondamentale funzione ecologica di questi insetti impollinatori per l'intero ecosistema.
Un'indagine di campo del Centro di referenza nazionale per l'apicoltura dell'Istituto Zooprofilattico Sperimentale delle Venezie nell'ambito di alcune morie riscontrate ha rilevato la presenza, in campioni di api morte, di residui di pesticidi e di alcuni virus delle api. Le infezioni virali potrebbero peggiorare l'impatto già negativo dei pesticidi sulla salute delle api, mettendo ulteriormente in pericolo la sopravvivenza delle colonie. Lo studio è stato effettuato su 94 campioni, provenienti dal Nord-est dell'Italia e raccolti durante la primavera 2014, prendendo in considerazione 150 principi attivi e 3 virus delle api. Lo studio è pubblicato su Journal of Apicultural Research. I ricercatori hanno riscontrato la presenza di almeno un princi-

pio attivo nel 72,2% dei campioni (api morte). Gli insetticidi sono i più abbondanti (59,4%), principalmente quelli appartenenti alla classe dei neonicotinoidi (41,8%), seguiti da fungicidi (40,6%) e acaricidi (24,1%). Gli insetticidi più frequentemente rilevati sono rappresentati da imidacloprid, chlorpyrifos, tau-fluvalinate e cyprodinil.
La presenza di una possibile relazione tra la mortalità primaverile delle api e l'impiego di trattamenti antiparassitari in agricoltura potrebbe contribuire a comprendere meglio fenomeni complessi come la moria delle api e lo spopolamento degli alveari, che negli ultimi dieci anni hanno colpito questo settore[2].
Lo scopo della presente tesi è quello di illustrare la progettazione di un software che visualizza come le molecole di specifici pesticidi si dispongono, in maniera spaziale, quando sono legate ai recettori delle api, questo processo viene definito **docking molecolare**, e successivamente il software estrae i legami che si vengono a formare.

Figura (1.1): Esemplare adulto di Apis Mellifera

1.1 Docking Molecolare

Il docking molecolare è una tecnica computazionale che mira a determinare le migliori conformazioni adottate da una molecola per legarsi ad un'altra al fine di formare un complesso stabile. Il docking molecolare viene fondamentalmente utilizzato per la valutazione delle interazioni ligando-target. A partire quindi da un ligando e dalla struttura nota del suo target, il docking molecolare permette di generare una serie di conformazioni possibili del ligando stesso localizzato all'interno del sito attivo della proteina. Esse sono denominate *"binding poses"*e sono valutate da particolari funzioni chiamate *"scoring functions"*, che creano quindi un vero e proprio ranking. Le *migliori poses* rappresentano quella che viene identificata come la miglior soluzione proposta dall'algoritmo per l'interazione tra il ligando e il target[3].
Gli algoritmi di docking sono formati da due componenti fondamentali: l'algoritmo di ricerca (o *"search algorithm"*) e la *"scoring function"*. Il primo si occupa di generare un insieme di 12 conformazioni del ligando all'interno del sito designato del target, mentre la seconda valuta le *poses* generate, assegnando a ciascuna di esse un punteggio (detto *"score"*) in base a parametri di tipo geometrico ed energetico. Le migliori conformazioni in uscita da questa valutazione sono passate nuovamente all'algoritmo di ricerca, che andrà a creare una nuova generazione di conformazioni partendo dalle migliori soluzioni della run precedente. Il funzionamento iterativo del *search algorithm* e della *scoring function* permettono di otte-

nere, alla fine di un determinato numero di cicli, un insieme di *poses* che vengono fornite come output all'utente e che sono ritenute essere le migliori soluzioni per il *binding* delle molecole in esame da parte del programma di docking utilizzato.
In generale, il docking molecolare è eseguibile in tre differenti condizioni, che si differenziano l'una dall'altra per i gradi di libertà tenuti in considerazione dall'algoritmo durante il calcolo:

- docking a corpo rigido, che approssima sia il ligando che la proteina come strutture rigide
- docking semi-flessibile, che considera il target come rigido, tendendo però in considerazione i gradi di libertà conformazionale del ligando
- docking flessibile, in cui vengono considerati i gradi di libertà sia del ligando che dei residui del target nel sito attivo.

Intuitivamente, passando da un approccio a corpo rigido fino ad uno flessibile, la complessità di calcolo aumenta, e, proporzionalmente, anche il tempo di esecuzione.
Ad oggi sono disponibili diversi protocolli di docking, e ognuno sfrutta una particolare coppia algoritmo di *ricerca-scoring function*.
Il docking molecolare consiste in tre obiettivi principali collegati tra loro: predizione della posa, screening virtuale e stima dell'affinità di legame. Una metodologia di docking di successo deve essere in grado di prevedere correttamente la posa

nativa del ligando all'interno del sito di legame del recettore (cioè di trovare la geometria sperimentale del ligando entro un certo limite di tolleranza) e le interazioni fisico-chimiche molecolari associate. Inoltre, quando si analizzano librerie di composti di grandi dimensioni, il metodo deve essere in grado di distinguere con successo le molecole che si legano da quelle che non si legano e di classificare correttamente questi ligandi tra i migliori composti del database. Un algoritmo di ricerca e una funzione di score energetico sono gli strumenti di base di una metodologia di docking per generare e valutare le conformazioni dei ligandi. La capacità di gestire con successo la flessibilità molecolare intrinseca di un sistema e di descrivere correttamente l'energia delle interazioni recettore-ligando è cruciale per lo sviluppo di metodologie di docking predittivo che sono utili negli studi prospettici di progettazione di farmaci[4].

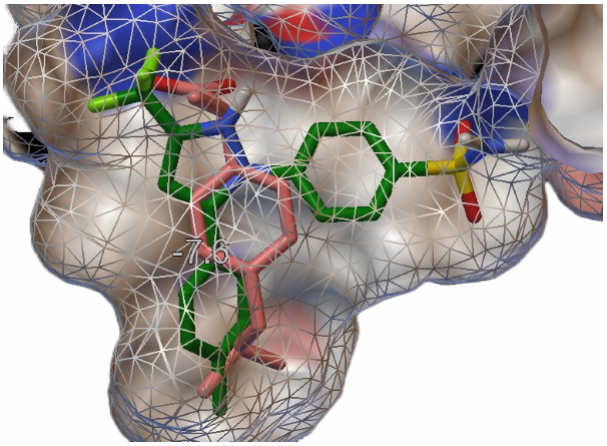


Figura (1.2): Rappresentazione dei due ligandi ibuprofene a sinistra e celecoxib a destra che hanno effettuato il docking con un enzima di COX2

1.2 Simulazione

La simulazione di un processo di docking è un processo molto più che complicato. In tale approccio, la proteina e il ligando sono separati fisicamente da una certa distanza, e il ligando trova la sua posizione nel sito attivo della proteina dopo aver compiuto diversi movimenti nello spazio. Tali movimenti includono rotazioni, traslazioni e torsione di alcuni angoli di rotazione degli atomi. Ognuno di questi movimenti ha un determinato costo energetico nel sistema, quindi dopo ogni mossa viene ricalcolata l'energia totale del sistema. Questo approccio modellizza molto precisamente quello che accade nella realtà. Di contro, il costo richiesto in termini di tempo e prestazioni è molto elevato.

1.3 Software per il docking molecolare

I programmi di docking molecolare eseguono un algoritmo di ricerca in cui la conformazione del ligando viene valutata ricorsivamente fino a raggiungere la convergenza all'energia minima. Infine, una funzione di punteggio di affinità, $\triangle$G (Energia potenziale totale in kcal/mol), viene impiegata per classificare le pose candidate come la somma delle energie elettrostatiche e di van der Waals. Le forze trainanti per queste specifiche interazioni nei sistemi biologici mirano alla complementarità tra la forma e l'elettrostatica delle superfici del sito di legame e del ligando o del substrato.
Negli ultimi vent'anni, sono stati sviluppati più di 60 diversi

strumenti e programmi di docking sia per uso accademico e commerciali, come DOCK (Venkatachalam et al. 2003) AutoDock (Österberg et al. 2002), FlexX (Rarey et al. 1996), Surflex (Jain 2003), GOLD (Jones et al. 1997), ICM (Schapira et al. 2003), Glide (Friesner et al. 2004), Cdocker, LigandFit (Venkatachalam et al. 2003), MCDock, FRED (McGann et al. 2003), MOE-Dock (Corbeil et al. 2012), LeDock (Zhao e Caflisch 2013), AutoDock Vina (Trott e Olson 2010), rDock (Ruiz-Carmona et al. 2014), UCSF Dock (Allen et al. 2015) e molti altri.

Tra questi programmi, AutoDock Vina, GOLD e MOE-Dock hanno predetto le pose migliori con gli score migliori. AutoDock e LeDock sono stati in grado di identificare i corretti legami dei ligandi nelle pose. Sia Glide (XP) che AutoDock hanno previsto le pose con un'accuratezza del 90,0% (Wang et al. 2016). È stato dimostrato che AutoDock ha prodotto fattori di arricchimento più rispetto a Glide in uno studio di screening virtuale contro il Fattore Xa, mentre Glide ha superato AutoDock contro lo stesso bersaglio in un analogo studio di screening virtuale. Nel complesso, è stato riportato recentemente che questi programmi di docking sono in grado di predire pose sperimentali con deviazioni al quadrato della radice (RMSD) in media (RMSD) in media da 1,5 a 2 Å[5].

Come mostrato nella Figura 1.3, il software di docking molecolare può aiutarci a individuare la conformazione e l'orientamento ottimali in base alla complementarità e alla preorganizzazione con un algoritmo specifico, quindi ad applicare una funzione di scoring per prevedere l'affinità del legame e

ad analizzare la modalità interattiva[6].

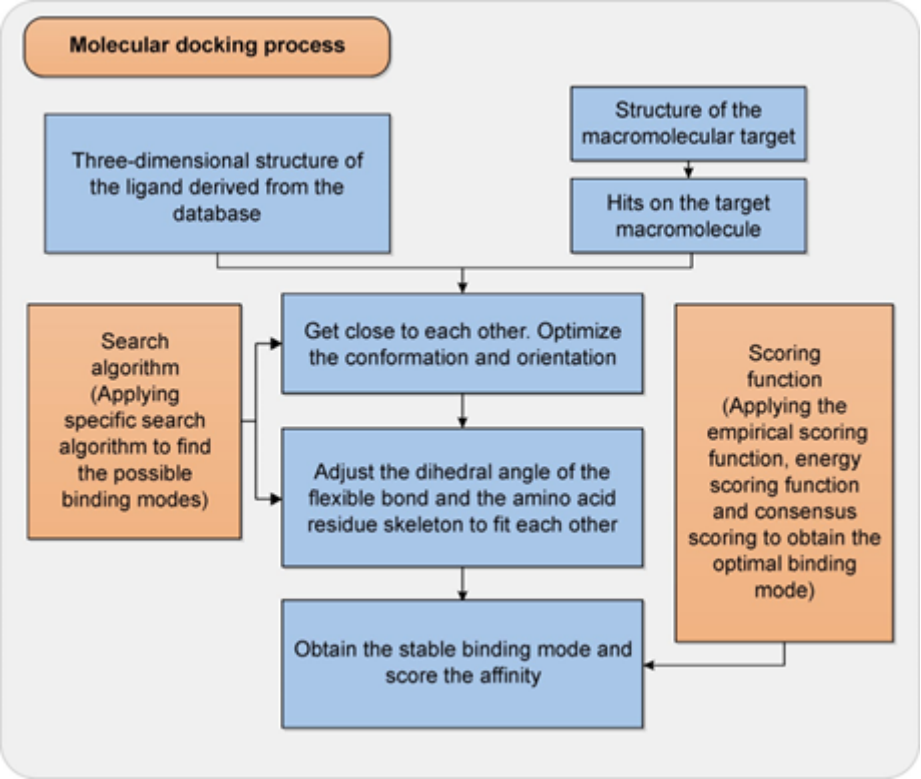


Figura (1.3): Processo del docking molecolare

1.4 Idea e sviluppo

L'**idea** nasce dall'attività di tirocinio svolta presso il "Centro nazionale di ricerca" di Napoli, per un totale di 300 ore (12 CFU), sotto la supervisione del Responsabile del laboratorio professore Angelo Ciaramella e del dottore Ferdinando Febbraio del CNR di Napoli. Il lavoro effettuato è consistito nella realizzazione di un software, del tutto preliminare al progetto di tesi proposto. Lo **sviluppo** è avvenuto attraverso diverse fasi nelle quali sono stati utilizzati ed implementati i seguenti tools:

- software per l'esecuzione del docking
- funzioni di bioinformatica per la preparazione degli input necessari
- software per l'analisi dei risultati dell'intero processo.

Sono state determinate le componenti software ideali per automatizzare il processo di docking conseguendo risultati efficienti per quanto riguarda l'output e l'analisi dello stesso, offrendo una buona usabilità del prodotto realizzato mediante una semplice ed intuitiva interfaccia grafica.

1.5 Contenuto della tesi

La tesi è divisa in tre moduli:

- nella prima parte verranno discusse le tecnologie, le piattaforme scelte per la realizzazione del software, i linguaggi di programmazione e gli strumenti di bioinformatica utilizzati
- nella seconda verrà illustrata l'applicazione realizzata, dalla preparazione dei ligandi e recettori, passando per il docking, finendo con l'estrazione dei legami dall'output ottenuto
- nell'ultima parte verranno tratte le conclusioni e saranno indicati gli sviluppi futuri del software realizzato.

Capitolo 2

Applicazione realizzata

Il progetto di tesi proposto ha come focus principale la realizzazione del docking tra i ligandi contenuti in specifici pesticidi e i recettori dell'apis mellifera e l'estrazione dei legami che si vengono a formare.

2.1 Installazione

L'applicazione viene scaricata dalla repository di github mediante il comando:

```
git clone https://github.com/mungowz/Computational-Docking.git
```

Verrà quindi installata nella directory corrente la repository contenente il progetto, all'interno di questa si trovano gli script utilizzati dall'applicazione e le directory ed i file di input di default, tra cui:

- Il file di input dei ligandi, ovvero *ligands_list.txt*
- La directory contenente la lista di default dei ligandi, ovvero */data/files/*

- La directory di default che conterrà i file *.xlsx* di output prodotti, ovvero */output/excel_files*
- La directory di default che conterrà i file *.sdf* prodotti, ovvero */data/ligands/sdf/*
- Le directory di default che conterranno i file *.pdb* prodotti, ovvero */data/ligands/pdb/* per i ligandi e */data/proteins/pdb/* per i recettori
- Le directory di default che conterranno i file *.pdbqt prodotti*, ovvero */data/ligands/pdbqt/* per i ligandi e */data/proteins/pdbqt/* per i recettori.

Sarà necessario inoltre installare alcune dipendenze esterne ed interne per far funzionare l'applicazione, la lista delle dipendenze e la procedura di installazione è specificata nel file *README* della repository di github.

2.2 Modalità di utilizzo

Il software realizzato può essere utilizzato in due modalità: mediante script python da terminale o mediante interfaccia grafica realizzata seguendo il paradigma *model-view-controller*. Non sarà necessario effettuare ulteriori installazione per usufruire di entrambe le modalità ma bisognerà semplicemente scaricare la repository e seguire le istruzioni del *README*.
Per eseguire l'applicazione da riga di comando devono essere eseguiti separatamente nell'ordine i seguenti script:

- **prepare_ligands.py**

- **prepare_receptors.py**
- **performDocking.sh**

La GUI usufruirà di versione modificate degli script lanciati da riga di comando il che garantisce le stesse funzionalità in entrambe le modalità di utilizzo.
Per avviare la GUI deve essere digitato nel terminale il comando:

```
python app.py
```

Avviando la GUI si aprirà la home page con le seguenti opzioni:

- Opzione **Preparation**: si accede alla sezione relativa alla preparazione degli input
- Opzione **Docking**: sarà possibile effettuare il docking.

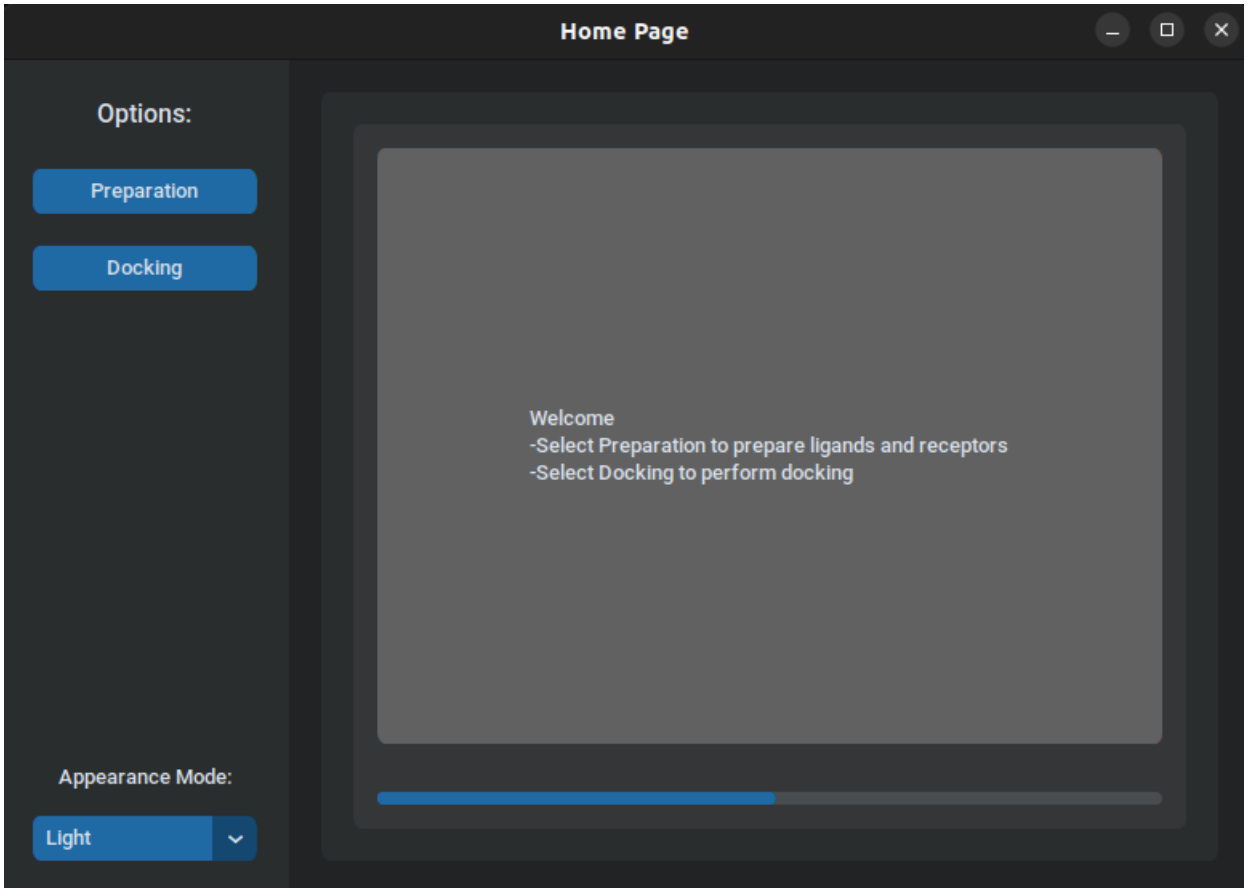


Figura (2.1): Sezione **Home Page** della GUI

2.3 Dati in input

I dati in input all'applicazione trattata sono: ligandi e proteine. La lista dei ligandi è fornita in input tramite foglio calcolo (*.xlsx*, *.xls*) o mediante file di testo (*.txt*), in entrambi i casi ogni riga corrisponde al nome di un ligando. Essendo l'applicazione incentrata sullo studio degli effetti dei ligandi dei pesticidi sull'apis mellifera, come dati di esempio sono state utilizzati i ligandi le cui molecole costituiscono i pesticidi maggiormente diffusi sul mercato, per un totale di 297 ligandi. La lista è disponibile nell'appendice A.1.

I recettori vengono selezionati mediante una **web view** aperta sulla pagina di ricerca del sito di PubChem, quivi l'utente digiterà la propria query e dopo aver selezionato il tasto **research** gli verranno mostrati tutti i composti organici relativi alla query digitata, tramite il tasto **Get Query** l'utente andrà a scaricare tutti i file dei composti organici in formato *.pdb*. Nel caso di esempio sono state scelte tutte le proteine dei recettori dell'Apis Mellifera come mostrato nelle foto: 3.2 e 3.3, per un totale di 60 file *.pdb* contenenti le strutture di determinate proteine. La lista delle proteine scaricate è disponibile nell'appendice: B.1.

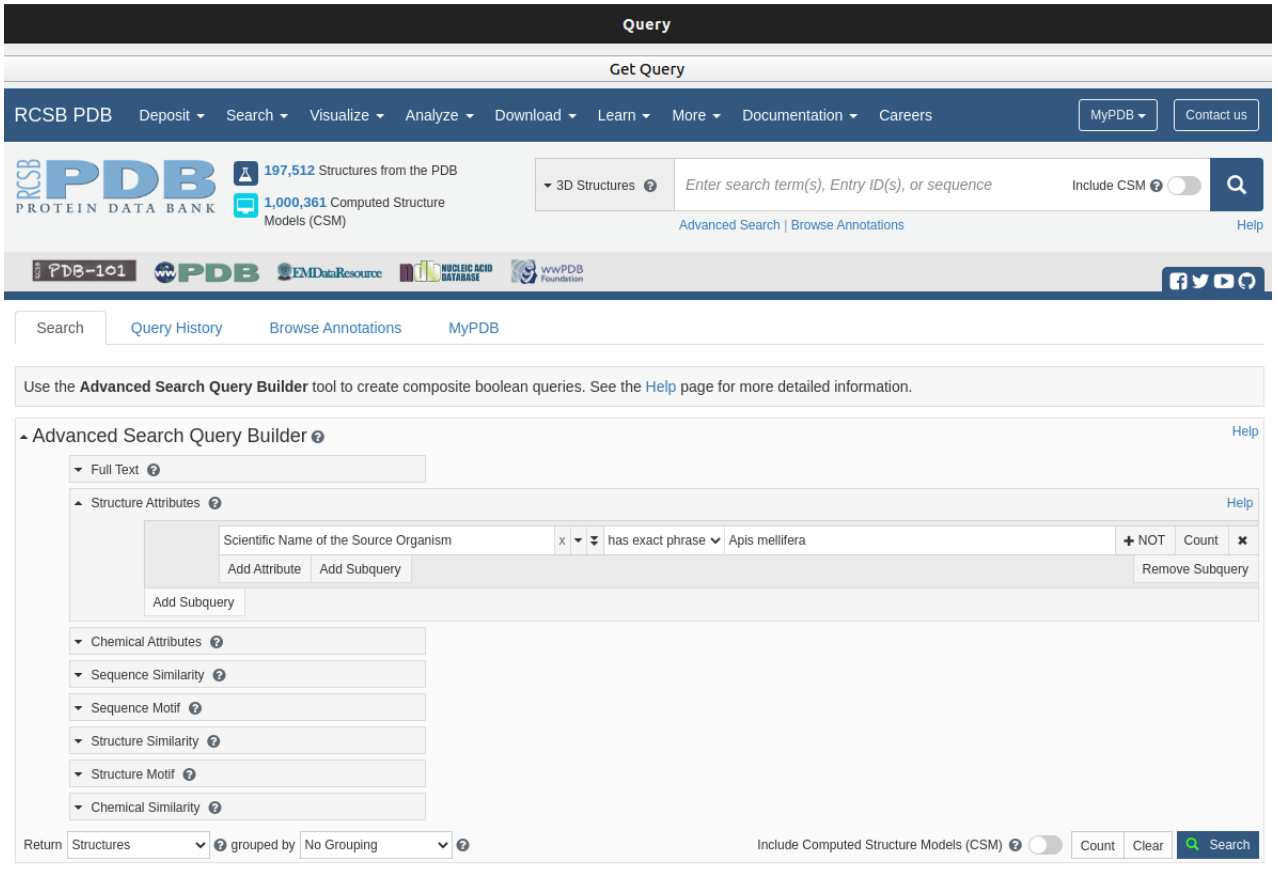


Figura (2.2): Query dei recettori

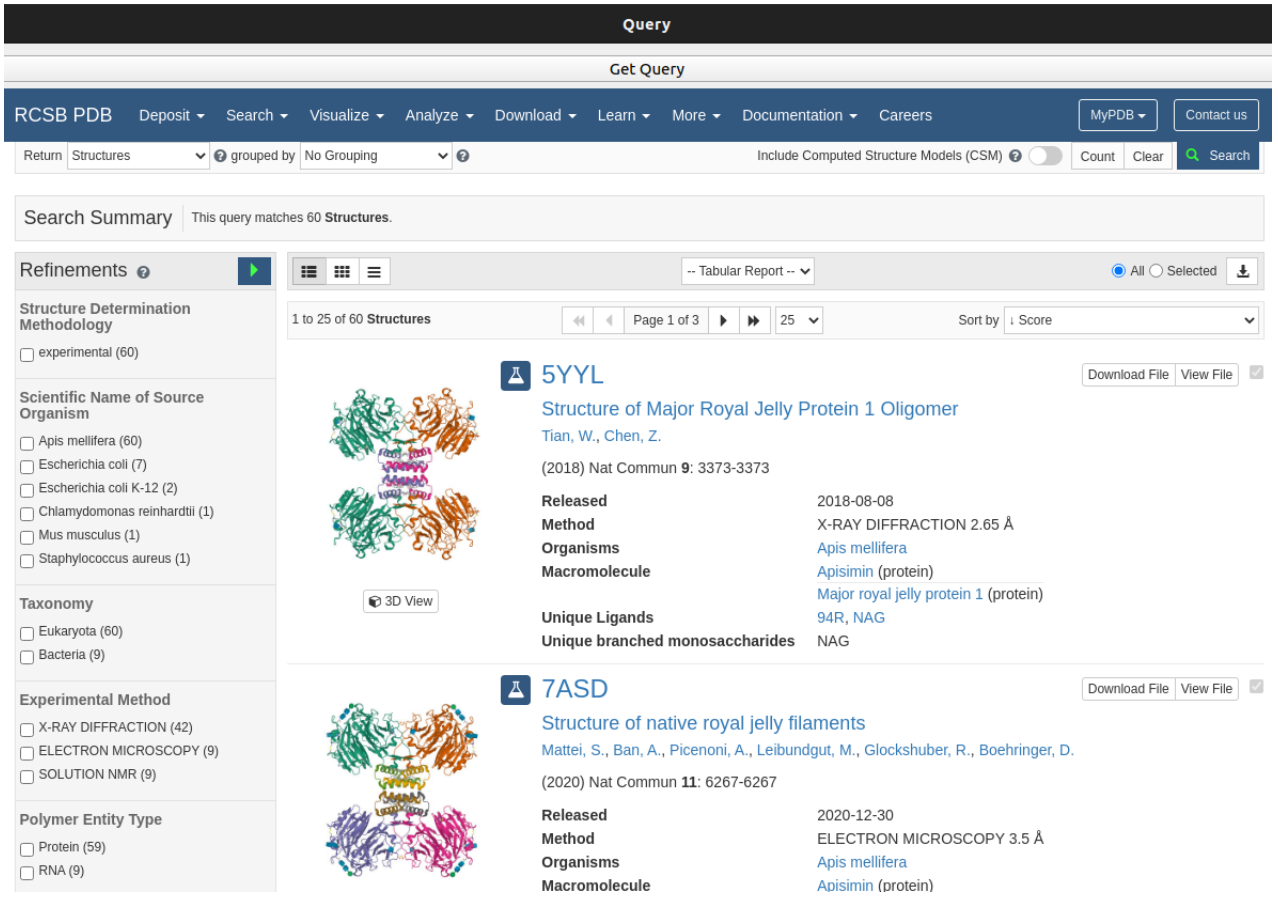


Figura (2.3): Proteine dei recettori dell'apis mellifera

2.4 Preparazione dei ligandi e dei recettori

Il primo step propedeutico per il docking è la preparazione dei ligandi e dei recettori, questa fase viene esplicitamente eseguita dal software realizzato. L'intera fase è svolta: per i ligandi dallo script python **prepare_ligands.py** mentre, per i recettori dallo script python **prepare_receptors.py**. Se l'applicazione viene utilizzata mediante GUI la preparazione dei ligandi e dei recettori può essere effettuata mediante la sezione **Preparation** della pagina principale come mostrato in

figura 3.4.

- Selezionando l'opzione **Ligands** si accede alla sezione relativa alla preparazione dei ligandi
- Selezionando l'opzione **Receptors** si accede alla sezione relativa alla preparazione dei recettori
- Selezionando l'opzione **Back** si ritorna alla pagina principale.

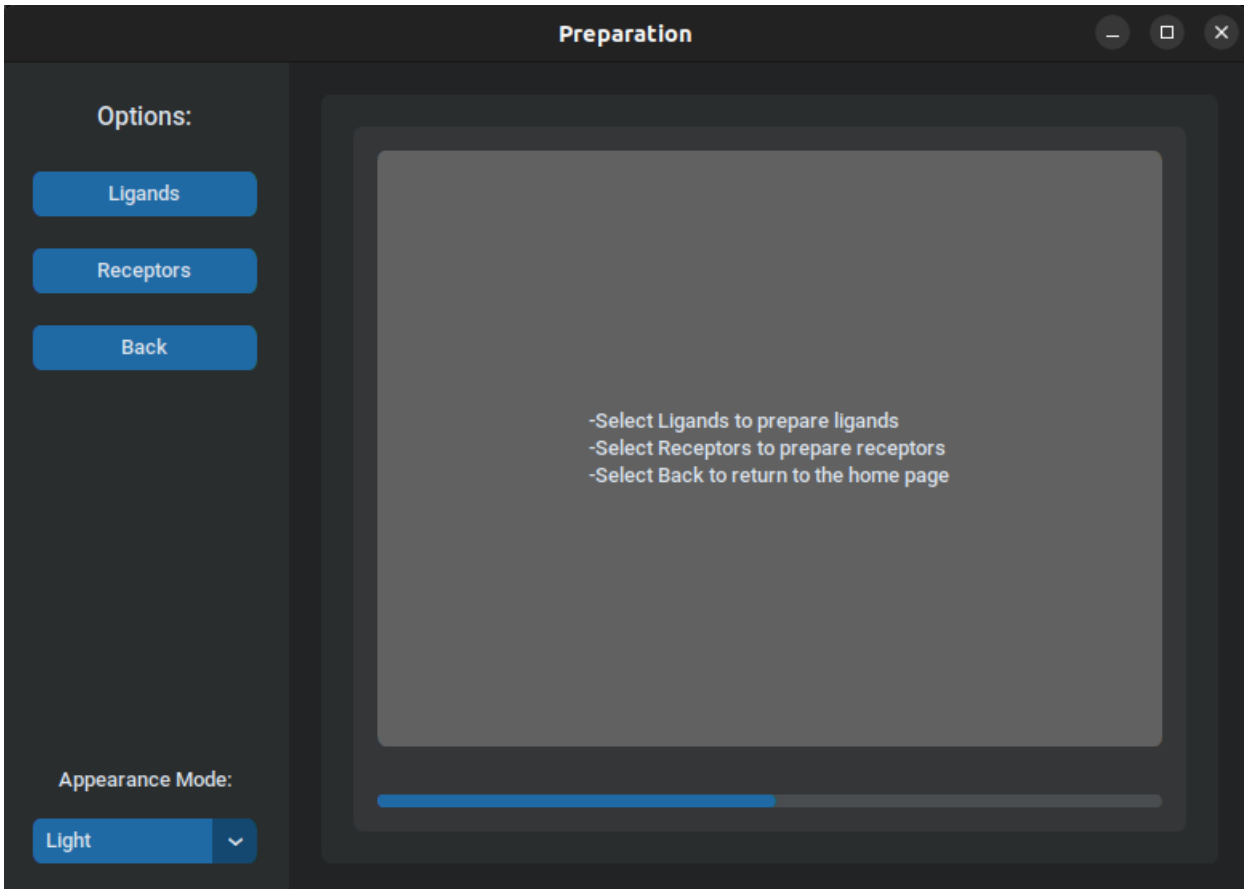


Figura (2.4): Sezione **Preparation** della GUI

2.4.1 Preparazione dei ligandi tramite script

Lo script python **prepare_ligands.py**, che esegue tale fase, viene eseguito da terminale mediante il comando:

```
python prepare_ligands.py
```

Questo script può ricevere diversi argomenti in input:

- $[-v]|[--verbose]$: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- $[-e]|[--input-file]$: specifica un nuovo file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi, se non viene specificata tale opzione verrà scelto il file di default scaricato insieme al software
- $[-E]|[--excel-folder]$: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-s]|[--sdf-folder]$: specifica la directory dei file *.sdf* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-P]|[--pdb-folder]$: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-p]|[--pdbqt-folder]$: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default

- $[-k]|[--keep-ligands]$: specificando questa opzione viene scelto di non cancellare i file dei ligandi precedentemente scaricati, se non viene specificata tale opzione i file verranno cancellati
- $[-h]|[--help]$: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, e nel caso siano stati inseriti in input verrà controllata l'esistenza e la validità degli stessi. Nel caso non siano presenti le directory contenenti i file *.sdf*, *.pdb* e *.pdbqt*, queste verranno create automaticamente dall'applicazione a meno di input inseriti dall'utente e saranno cancellati i file precedentemente scaricati a meno di input contrario.
Lo script richiama la funzione **selectLigands** la quale si occupa di scaricare da **Pubchem** i file *.sdf* corrispondenti alla lista dei ligandi in input. La funzione prende in input:

- *input_path*, il path del file di input
- *sdf_folder*, il path della directory di output per i file *.sdf*
- *excel_folder*, il path della directory di output per i file *.xlsx*
- *verbose*, il flag relativo al verbose.

In output ci vengono dati i file *.sdf* corrispondenti ai ligandi in input, dove il nome di ogni file è preceduto dal suffisso *ligand_*, e un file *.xlsx* contenente i risultati dell'operazione di download nominato *ligands_sdf_output.xls*.

```
selectLigands(input_path, sdf_folder, excel_folder, verbose)
```

L'interfaccia con il database avviene tramite il pacchetto **PubChemPy**, in particolare la funzione **pubchempy.get_compounds** permette di ricercare nel database il record relativo allo specifico ligando il cui nome, a cui viene precedentemente aggiunto il suffisso *ligand_*, gli viene fornito in input. Questa prende in input:

- *identifier*, il composto da ricercare nel database
- *namespace*, il parametro in base al quale ricercare il composto, nel nostro caso il parametro scelto è il nome indicato dalla stringa *"name"*
- *record_type* ovvero il tipo di record relativo al composto in questione da scaricare, nel nostro caso il parametro scelto è la stringa *"3d"* che indica la struttura 3D del ligando.

In output verrà fornito il record della struttura 3D del ligando in questione.

```
pubchempy.get_compounds(identifier, "name", record_type="3d")
```

La funzione che effettivamente si occupa di scaricare la struttura 3D del ligando in formato *.sdf*, richiamata da **selectLigands** è **pubchempy.download**, prende in input:

- *outformat*, il formato in cui deve essere scaricato il file, nel nostro caso il parametro scelto è il formato *.sdf* indicato dalla stringa *"SDF"*
- *path*, il path della directory in cui vengono scaricati i file *.sdf*

- *identifier*, il composto da scaricare nel database
- *namespace*, il parametro in base al quale ricercare il composto, nel nostro caso il parametro scelto è il nome indicato dalla stringa *"name"*
- *record_type* ovvero il tipo di record relativo al composto in questione da scaricare, nel nostro caso il parametro scelto è la stringa *"3d"* che indica la struttura 3D del ligando.

L'output di tale funzione sarà il file *.sdf* del ligando in questione.

```
pubchempy.download("SDF", path, identifier, "name", record_type="3d")
```

I nomi di alcuni file potrebbero non essere presenti nel database oppure, a causa di spazi presenti nel loro nome, non essere riconosciuti. Nel primo caso i ligandi vengono semplicemente scartati nel secondo caso gli spazi vengono sostituiti da underscore (_), seguendo quindi la nomenclatura del database, e viene effettuata nuovamente la ricerca di questi nel database. Se anche stavolta la ricerca non ha successo il ligando in questione viene definitivamente scartato.
Oltre ai file *.sdf* la **selectLigands** produrrà anche un file *.xlsx* contenente il riepigolo dei risultati ottenuti dalla funzione, ovvero:

- file scaricati
- file scaricati con il nome modificato
- ligandi non trovati all'interno del database.

Nel caso di esempio, dei 297 ligandi in input, 289 sono stati trovati e scaricati con successo, i restanti 8 sono stati scartati: 3.1.

Sodium 4-nitrophenolate	Silthiofa
Sodium 2-methoxy-5-nitrophenolate	Potassium bicarbonate
(E,E)-7,9-Dodecadien-1-yl acetate	Dodine
Sodium 2-nitrophenolate	Ziram

Tabella (2.1): Ligandi non scaricati

Dopo aver scaricato i file *.sdf* questi devono essere convertiti in formato *.pdb*, per fare ciò lo script richiama la funzione **sdf2pdb**, la quale prende in input:

- *sdf_folder*, la directory con i file *.sdf* in input
- *pdb_folder*, la directory con i file *.pdb* in output
- *verbose*, il flag relativo al verbose.

La funzione restituisce in output i file *.pdb* corrispondenti a tutti i file *.sdf* dati in input.

```
sdf2pdb(sdf_folder, pdb_folder, verbose)
```

La funzione usufruisce del programma **Open babel** in particolare esegue la conversione da *.sdf* a *.pdb* richiamando la sua versione da terminale tramite il comando **obabel**:

```
obabel sdf_file_path -O pdb_path
```

Nell'istruzione sopra *sdf_file_path* indica la directory con i file *.sdf* in input, la directory con i file *.pdb* in output, *pdb_path*, viene specificata tramite lo switch *-O*. Tramite tale istruzione è possibile effettuare la conversione di un singolo file, la

conversione di tutti i *.sdf* avviene ciclando su tutti i file nella directory corrispondente.
L'ultimo step nella preparazione dei ligandi è effettuato dalla funzione **prepareLigands**, questa funzione prende in input:

- *pdb_folder*, la directory con i file *.pdb* in input
- *pdbqt_folder*, la directory con i file *.pdbqt* in output
- *verbose*, il flag relativo al verbose.

La funzione restituisce la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

```
prepareLigands(pdb_folder, pdbqt_folder, verbose)
```

prepareLigands usufruisce dello script **prepare_ligand4** offerto dalla suite **ADFR** tramite il quale effettua la corretta conversione da *.pdb* a *.pdbqt*. Lo script viene richiamato tramite riga di comando dalla funzione mediante il comando **prepare_ligand**:

```
prepare_ligand -l pdb_file_path -v -o pdbqt_path
```

Nell'istruzione sopra, *pdb_file_path* indica la directory con i file *.pdb* in input ed è specificata tramite lo switch *-l*, lo switch *-v* indica che è attivato il verbose e la directory con i file *.pdbqt* in output, *pdbqt_path*, viene specificata tramite lo switch *-o*. Tramite tale istruzione è possibile effettuare la conversione di un singolo file, la conversione di tutti i *.pdb* avviene ciclando su tutti i file nella directory corrispondente.

2.4.2 Preparazione dei ligandi tramite GUI

La preparazione dei ligandi tramite GUI avviene selezionando il tasto **Ligands** nella sezione **Preparation** del software. All'inteno della pagina relativa alla preparazione dei ligandi premendo il tasto **Execute** è possibile effettuare i procedimenti precedentemente spiegati nella sezione relativa all'esecuzione tramite script (3.4.1). Premendo il tasto **Back** è possibile tornare alla sezione **Preparation** relativa alla preparazione degli input. Come si osserva nella figura 3.5 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- un file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi
- la directory dei file *.xlsx* di output
- la directory dei file *.sdf* di output
- la directory dei file *.pdb* di output
- la directory dei file *.pdbqt* di output.

Se lasciate vuote queste entry verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale se selezionato permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.

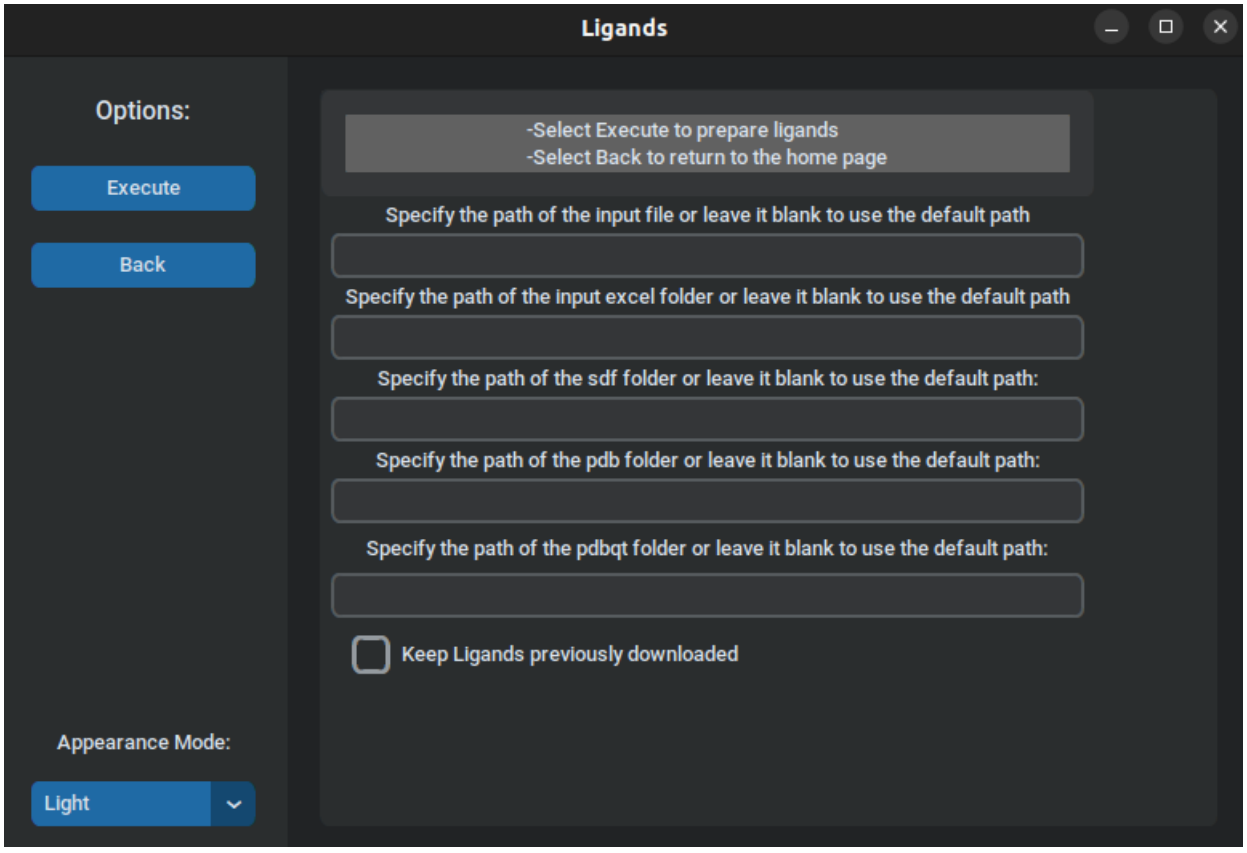


Figura (2.5): Sezione **Ligands** della GUI

La GUI per la preparazione dei ligandi richiama versioni modificate delle funzioni e dello script **prepare_ligands.py** discussi nella precedente sezioni, le modifiche apportate semplicemente adattano le procedure e gli script all'interfaccia grafica senza eliminare alcuna funzionalità precedentemente illustrata per la versione da riga di comando.
Durante l'esecuzione vengono mostrate diverse **progress bar** (figura: 3.6, 3.7 e 3.8) in modo tale che sia restituito un feedback all'utente relativo al progresso della preparazione dei ligandi. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.9) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.10).

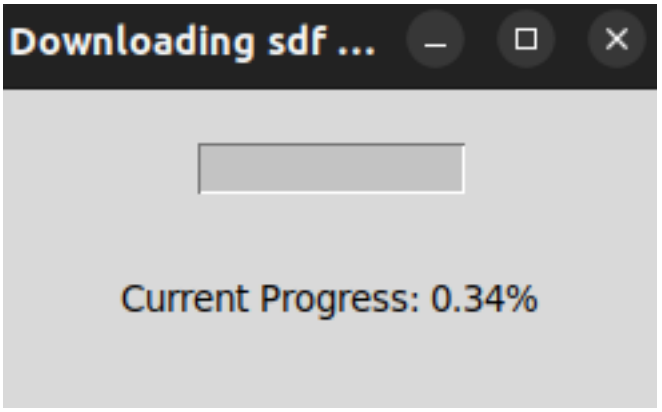


Figura (2.6): **Progress bar** dei file *.sdf*

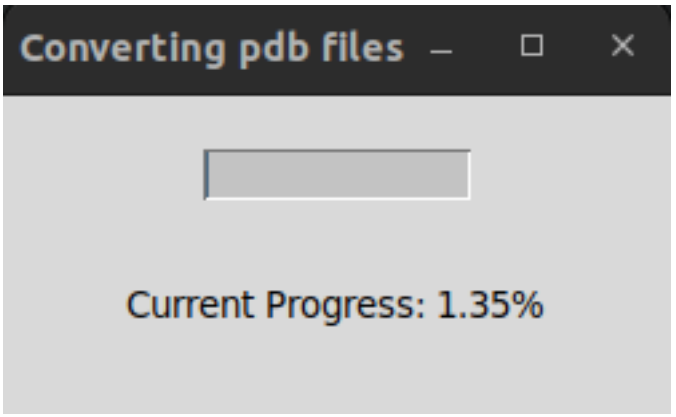


Figura (2.7): **Progress bar** dei file *.pdb*

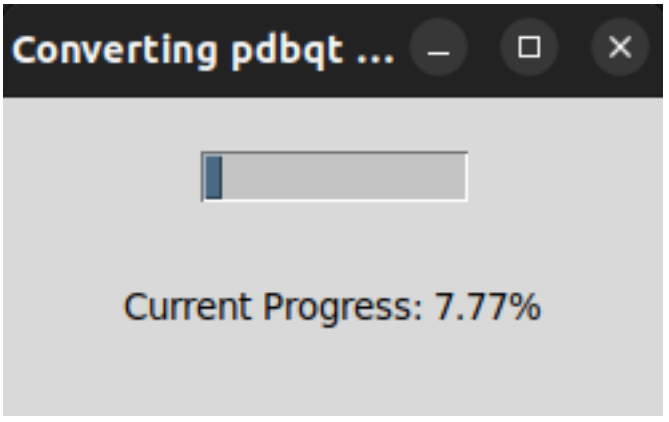


Figura (2.8): **Progress bar** dei file *.pdbqt*

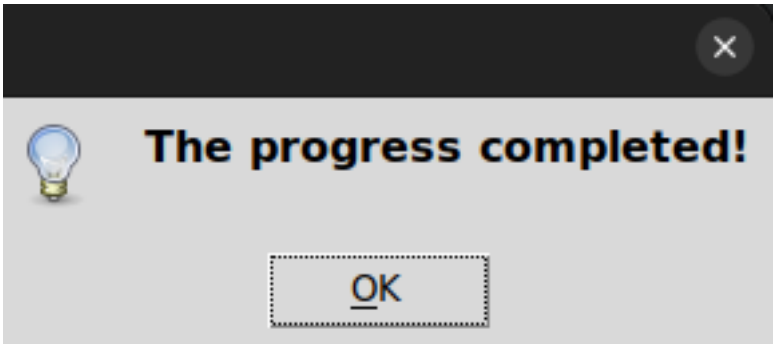


Figura (2.9): Messaggio processo completato con successo

Figura (2.10): Messaggio di errore

2.4.3 Preparazione dei recettori tramite script

o script python **prepare_receptors.py**, che esegue tale fase, viene eseguito da terminale mediante il comando:

```
python prepare_receptors.py
```

Questo script può ricevere diversi argomenti in input:

- $[-v]|[--verbose]$: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- $[-E]|[--excel-folder]$: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default

- $[-P]|[--pdb-folder]$: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-p]|[--pdbqt-folder]$: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-k]|[--keep-pdb-files]$: specificando questa opzione viene scelto di non cancellare i file delle proteine precedentemente scaricati, se non viene specificata tale opzione i file verranno cancellati
- $[-m]|[--margin]$: specificando questa opzione viene fornita in input la dimensione in *angstrom* del margine della *grid box* per le proteine se non viene specificata tale opzione viene impostato il valore di default ovvero 3
- $[-h]|[--help]$: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, e nel caso siano stati inseriti in input verrà controllata l'esistenza e la validità degli stessi. Nel caso non siano presenti le directory contenenti i file *.txt*, *.pdb* e *.pdbqt*, queste verranno create automaticamente dall'applicazione a meno di input inseriti dall'utente e saranno cancellati i file precedentemente scaricati a meno di input contrario.
Lo script richiama la funzione **selectReceptors** la quale si occupa di scaricare da **Pubchem** i file *.pdb* corrispondenti alle proteine scelte. La funzione prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *excel_folder*, il path della directory di output per i file *.xlsx*
- *verbose*, il flag relativo al verbose.

In output ci vengono dati i file *.pdb* corrispondenti ai recettori scelti in input e un file *.xlsx* contenente i risultati dell'operazione di download nominato *info_proteins.xls*.

```
selectReceptors(pdb_folder, excel_folder, verbose)
```

All'interno di questa **selectReceptors** viene richiamata la funzione **RestApiSelection** la quale prende in input l'*URL* di PubChem:

```
RestApiSelection(URL)
```

Questa funzione apre una **web view** sul sito di PubChem come mostrato nella figura 3.2, l'utente potrà digitare la propria query e scaricare i composti scelti selezionando il tasto in alto **Get Query**.
La **web view** restituisce l'**URL** della pagina del database contenente i composti selezionati dalla query, il *JSON* di tale pagina viene estratto e convertito nella struttura dati *dizionario* di python tramite la funzione **loads** della libreria **json**.

```
dictionary = json.loads(data)
```

Nel *dizionario* restituito dalla funzione sarà contenuto l'elenco dei composti selezionati tramite la query.
Il software rieseguirà direttamente la query tramite la funzione **get** contenuta nella libreria **requests**. Questa funzione permette di inviare una richiesta di tipo *HTTP/1.1* prendendo in

input **URL** della query al quale viene aggiunto l'elenco di composti da scegliere in formato *JSON*, ciò eseguito convertendo il *dizionario* precedentemente ottenuto in **JSON** mediante la funzione **dump** della libreria **json**:

```
dictionary = json.dump(dictionary)
```

La funzione **get** restituisce la lista di proteine da scaricare:

```
requests.get(f"https://search.rcsb.org/rcsbsearch/v2/query?json={dictionary}")
```

La funzione **selectReceptors** richiama successivamente la funzione **downloadPdbs** che prende in input:

- *pdb_list*, la lista di proteine precedentemente ottenuta
- *outputh_path*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

La funzione ritorna in output i file *.pdb* delle proteine contenute nella lista in input:

```
downloadPdbs(pdbs_list, output_path, verbose)
```

La funzione **downloadPdbs** richiama la funzione **fetchPDB** del pacchetto **ProDy**, questa prende in input:

- *protein_code*, la proteina da scaricare
- *folder*, il path della directory di output per i file *.pdb*, nel nostro caso sarà impostato ad *out_path*

- *compressed*, il flag per decidere se il file scaricato dovrà essere compresso oppure no, nel nostro caso il flag sarà impostato a *False* per indicare che i file scaricati debbano essere già decompressi.

Tramite tale istruzione è possibile interfacciarsi con **PubChem** e scaricare di un singolo file *.pdb* mediante richiesta *FTP*, il download di tutte le proteine della lista avviene ciclando su tutti gli elementi di essa:

```
fetchPDB(protein_code, folder=output_path, compressed=False)
```

Dopo aver effettuato il download di tutte i file *.pdb* le informazioni relative al download ed ai file scaricati vengono salvati in un file *.xlsx* chiamato *info_proteins.xlsx* salvato nella directory */output/*.
Lo script **prepare_receptors.py** una volta terminata la funzione **selectReceptors** richiama **splitRepeatedResidues** che prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di rimuovere i residui ripetuti memorizzati nel file *.pdb* sotto la dicitura *alternative location* o *alt_loc*:

```
splitRepeatedResidues(pdb_folder, verbose, output_folder=None)
```

Per accedere agli attributi dei file *.pdb* utilizziamo la funzione **read_pdb** del pacchetto **PandasPdb**, questa prende in input il nome del file *.pdb*:

```
PandasPdb.read_pdb(pdb_file)
```

In particolare gli attributi *A* e *B* sono quelli ripetuti e i *B* saranno quelli eliminati nella colonna *alt_loc*.
Successivamente viene richiamata dallo script la funzione **deleteHeteroatomsChains**, la quale prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di rimuovere le catene di eteroatomi contenute nel file *.pdb*, queste sono evidenziate nel file tramite un loro attributo ovvero la keyword *HETATM*, una volta rimosse, le catene vengono salvate.

```
deleteHeteroatomsChains(pdb_folder, verbose)
```

L'estrazione degli attributi dai file *.pdb* avviene anche in questo caso tramite la funzione **read_pdb** precedentemente descritta.
prepare_receptors.py richiama successivamente la funzione **splitChains** che prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di dividere le catene di residui che formano la struttura della proteina mantenendo soltanto le catene distinte, cioè quelle catene che hanno solo una sequenza aminoacidica distinta e non ripetuta da altre catene.

```
splitChains(pdb_folder, verbose)
```

Tramite la funzione **parsePDB** della libreria **prody** viene ricostruita la struttura proteica attraverso oggetti come: la lista delle molecole, la lista degli atomi e dei residui, questa viene rappresentata da un oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**.

```
atoms = parsePDB(protein_file)
```

La ricostruzione della struttura proteica avviene tramite la vista gerarchica della struttura, ciò viene implementata tramite il metodo **getHierView** della classe **AtomGroup**:

```
atoms.getHierVieW()
```

Vengono quindi esaminate tutte le sequenze, sempre tramite la funzione **parsePDB**, e scelte solo le sequenze distinte, infine vengono salvate in un file tutte le catene distinte come se fossero delle proteine attraverso la funzione **writePDB** di **prody**, questa funzione prende in input:

- *filename*, il nome del file
- *new_atoms*, il composto da salvare

Questi file prenderanno il nome della proteina principale al quale viene aggiunto un underscore (_) seguito dal nome della catena distinta salvata nel file.

```
writePDB(filename, new_atoms)
```

Lo step successivo nella preparazione dei recettori è effettuato dalla funzione **prepareReceptors**, questa funzione prende in input:

- *pdb_folder*, la directory con i file *.pdb* in input
- *pdbqt_folder*, la directory con i file *.pdbqt* in output
- *verbose*, il flag relativo al verbose
- *charges_to_add*, la carica da aggiungere alle proteine.

La funzione restituisce la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

```
prepareReceptors(pdb_folder, pdbqt_folder, verbose)
```

prepareReceptors utilizza lo script **prepare_receptors4** offerto dalla suite **ADFR**. In realtà viene utilizzata una versione modificata di tale script: **replacePrepareReceptor4.sh**. Questa versione modificata tramite bash scripting sostituisce la carica di Gasteiger con la carica in input ovvero *charges_to_add* che nel nostro caso sono cariche di Kollman come specificate dalla stringa *"Kollman"*. Tramite questo script si effettua la corretta conversione da *.pdb* a *.pdbqt*. Lo script viene richiamato tramite riga di comando dalla funzione mediante il comando **prepare_receptor**:

```
prepare_receptor
-r
pdb_file_path
-A
checkhydrogens
-C
charges_to_add
-e
-o
pdbqt_folder
```

Nell'istruzione sopra:

- *pdb_file_path* indica la directory con i file *.pdb* in input ed è specificata tramite lo switch *-r*
- *checkydrogens* indica l'opzione tramite la quale vengono controllati gli atomi di idrogeno specificata lo switch *-A*
- *charges_to_add* indica la carica usata in input impostata tramite lo switch *-C*
- *pdbqt_folder* indica la directory con i file *.pdbqt* in output specificata tramite lo switch *-o*

Tramite tale istruzione è possibile effettuare la conversione di un singolo file, mentre la conversione di tutti i *.pdb* avviene ciclando su tutti i file nella directory corrispondente.
L'ultima fase nella preparazione dei ligandi consiste nella creazione delle *grid box*, mediante la funzione **createGridboxes** richiamata dallo script **prepare_receptors.py**, la funzione prende in input:

- **pdb_folder**, la directory contenente i file *.pdb*
- **gridbox_outputh_folder**, la directory di output per le *grid box*
- **margin**, la dimensione in *angstrom* del margine della *grid box*
- **verbose**, il flag relativo al verbose.

Per ogni proteina ottenuta deve essere creata una *grid box*, questa definisce lo spazio conformazionale dove si colloca la proteina, è definita da:

- Un centro
- Le dimensioni
- L'esaustività, la quale sarà utilizzata da qualsiasi software da **AutoDock Vina** per la ricerca delle pose possibili del ligando.

Una *grid box* definisce la regione delle proteine dove il docking viene effettuato. Qualsiasi altra regione al di fuori dalla *grid box* non verrà presa in considerazione durante il processo di docking.
Le *grid box* sono un input richiesto da **AutoDock Vina** ma non da altri software per il docking.

```
createGridboxes(pdb_folder, gridbox_output_folder, margin, verbose)
```

Per ogni proteina vengono estratti gli attributi contenuti nel proprio file e viene richiamata da **createGridboxes** la funzione **createGridbox**, questa prende in input:

- **ppdb**, l'insieme di attributi del file *.pdb* ottenuti come oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**
- **protein_path**, il file *.pdb* corrispondente ad una singola proteina
- **gridbox_outputh_folder**, la directory di output per le *grid box*
- **margin**, la dimensione in *angstrom* del margine della *grid box*

- **verbose**, il flag relativo al verbose.

La funzione **createGridbox** costruisce una *grid box* per una singola proteina utilizzando gli attributi del file *.pdb* rispettando i parametri precedentemente definiti per una *grid box*.

2.4.4 Preparazione dei reccettori tramite GUI

La preparazione dei recettori tramite GUI avviene selezionando il tasto **Receptors** nella sezione **Preparation** del software. All'inteno della pagina relativa alla preparazione dei recettori premendo il tasto **Execute** è possibile effettuare i procedimenti precedentemente spiegati nella sezione relativa all'esecuzione tramite script (3.4.3). Premendo il tasto **Back** è possibile tornare alla sezione **Preparation** relativa alla preparazione degli input. Come si osserva nella figura 3.11 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- un file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi
- la directory dei file *.xlsx* di output
- la directory delle *grid box* di output *.txt*
- la directory dei file *.pdb* di output
- la directory dei file *.pdbqt* di output
- il valore del margine delle *grid box*
- la carica da aggiungere.

Se lasciate vuote queste entry verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale se selezionato permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.

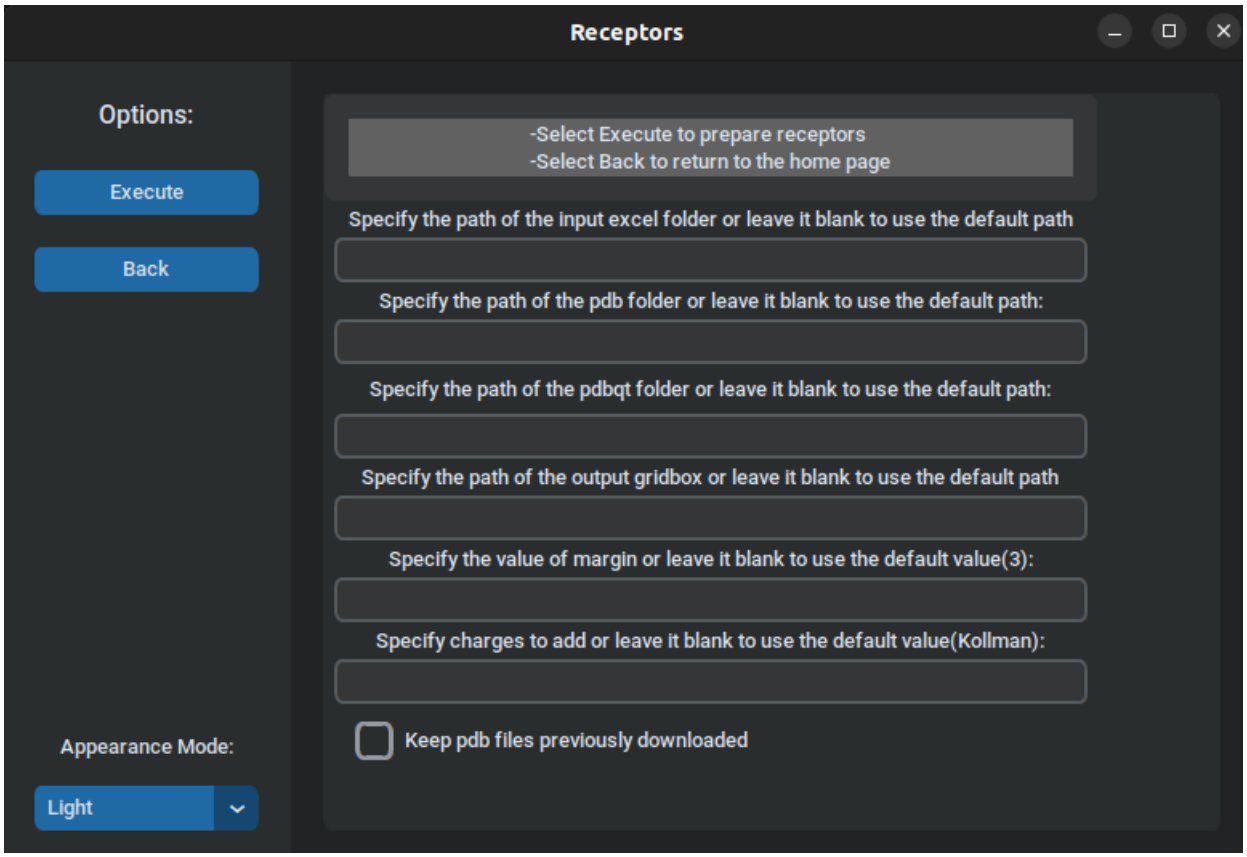


Figura (2.11): Sezione **Receptors** della GUI

La GUI per la preparazione dei recettori richiama versioni modificate delle funzioni e dello script **prepare_receptors.py** discussi nella precedente sezioni, le modifiche apportate semplicemente adattano le procedure e gli script all'interfaccia grafica senza eliminare alcuna funzionalità precedentemente illustrata per la versione da riga di comando.
Durante l'esecuzione viene mostrata una **progress bar** (figu-

ra: 3.12) in modo tale che sia restituito un feedback all'utente relativo al progresso della preparazione dei ligandi. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.13) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.14).

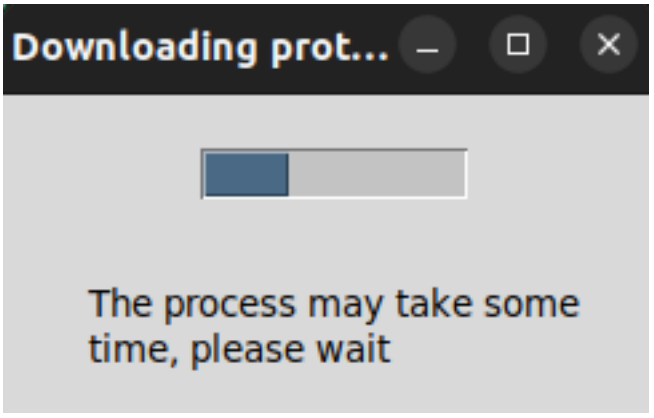


Figura (2.12): **Progress bar** delle proteine

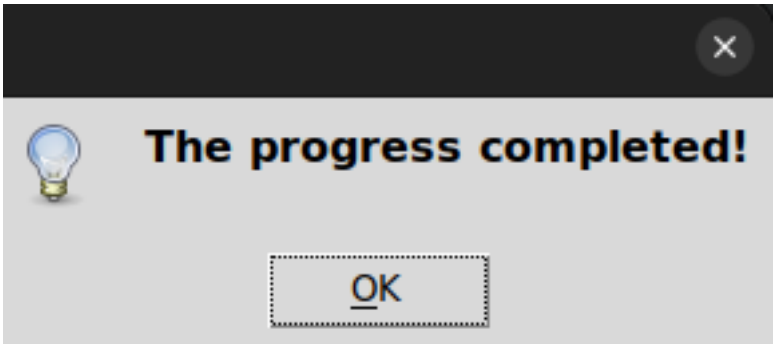


Figura (2.13): Messaggio processo completato con successo

Figura (2.14): Messaggio di errore

2.5 Docking

Capitolo 3

Applicazione realizzata

Il progetto di tesi proposto ha come focus principale la realizzazione del docking tra i ligandi contenuti in specifici pesticidi e i recettori dell'apis mellifera e l'estrazione dei legami che si vengono a formare.

3.1 Installazione

L'applicazione viene scaricata dalla repository di github mediante il comando:

```
git clone https://github.com/mungowz/Computational-Docking.git
```

Verrà quindi installata nella directory corrente la repository contenente il progetto, all'interno di questa si trovano gli script utilizzati dall'applicazione e le directory ed i file di input di default, tra cui:

- Il file di input dei ligandi, ovvero *ligands_list.txt*
- La directory contenente la lista di default dei ligandi, ovvero */data/files/*

- La directory di default che conterrà i file *.xlsx* di output prodotti, ovvero */output/excel_files*
- La directory di default che conterrà i file *.sdf* prodotti, ovvero */data/ligands/sdf/*
- Le directory di default che conterranno i file *.pdb* prodotti, ovvero */data/ligands/pdb/* per i ligandi e */data/proteins/pdb/* per i recettori
- Le directory di default che conterranno i file *.pdbqt prodotti*, ovvero */data/ligands/pdbqt/* per i ligandi e */data/proteins/pdbqt/* per i recettori.

Sarà necessario inoltre installare alcune dipendenze esterne ed interne per far funzionare l'applicazione, la lista delle dipendenze e la procedura di installazione è specificata nel file *README* della repository di github.

3.2 Modalità di utilizzo

Il software realizzato può essere utilizzato in due modalità: mediante script python da terminale o mediante interfaccia grafica realizzata seguendo il paradigma *model-view-controller*. Non sarà necessario effettuare ulteriori installazione per usufruire di entrambe le modalità ma bisognerà semplicemente scaricare la repository e seguire le istruzioni del *README*.
Per eseguire l'applicazione da riga di comando devono essere eseguiti separatamente nell'ordine i seguenti script:

- **prepare_ligands.py**

- **prepare_receptors.py**
- **performDocking.sh**

La GUI usufruirà di versione modificate degli script lanciati da riga di comando il che garantisce le stesse funzionalità in entrambe le modalità di utilizzo.
Per avviare la GUI deve essere digitato nel terminale il comando:

```
python app.py
```

Avviando la GUI si aprirà la home page con le seguenti opzioni:

- Opzione **Preparation**: si accede alla sezione relativa alla preparazione degli input
- Opzione **Docking**: sarà possibile effettuare il docking.

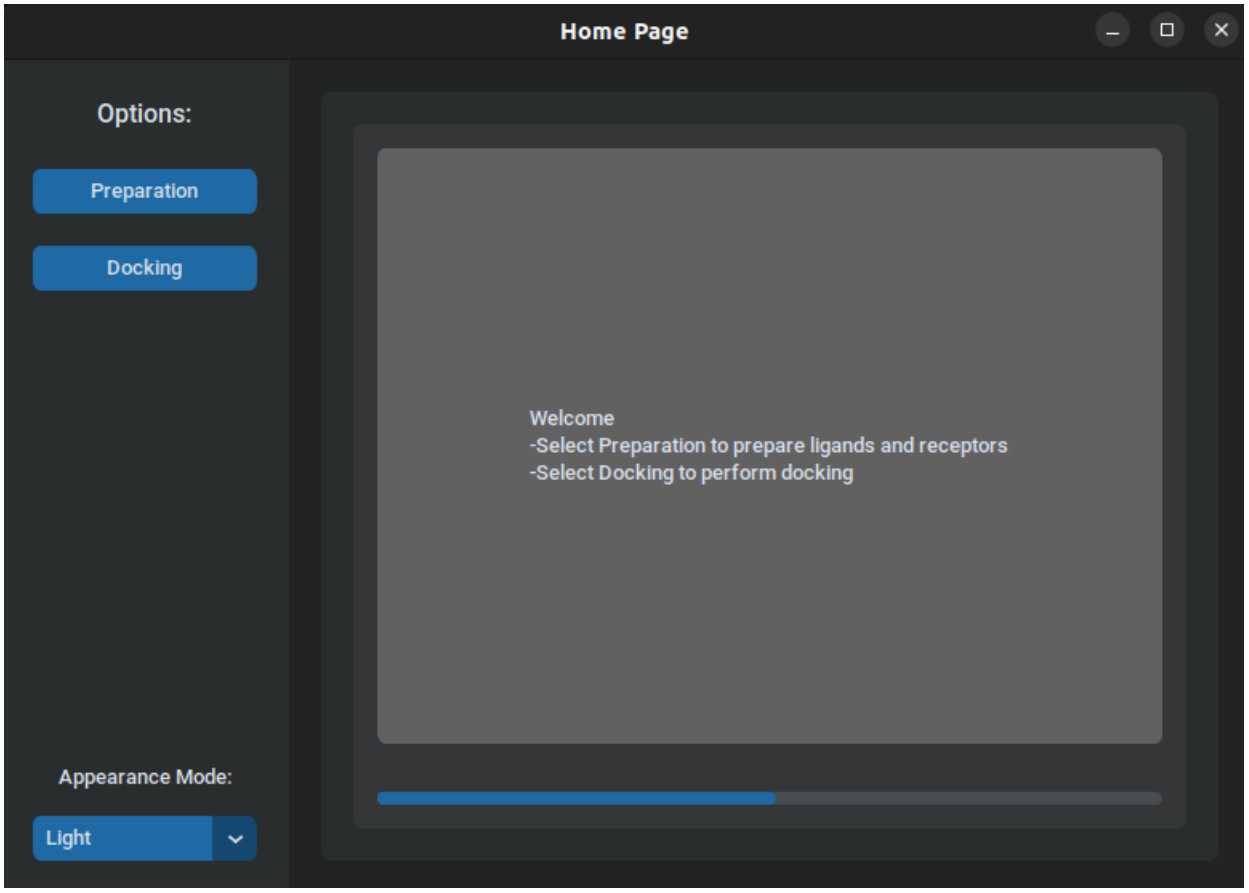


Figura (3.1): Sezione **Home Page** della GUI

3.3 Dati in input

I dati in input all'applicazione trattata sono: ligandi e proteine. La lista dei ligandi è fornita in input tramite foglio calcolo (*.xlsx*, *.xls*) o mediante file di testo (*.txt*), in entrambi i casi ogni riga corrisponde al nome di un ligando. Essendo l'applicazione incentrata sullo studio degli effetti dei ligandi dei pesticidi sull'apis mellifera, come dati di esempio sono state utilizzati i ligandi le cui molecole costituiscono i pesticidi maggiormente diffusi sul mercato, per un totale di 297 ligandi. La lista è disponibile nell'appendice A.1.

I recettori vengono selezionati mediante una **web view** aperta sulla pagina di ricerca del sito di PubChem, quivi l'utente digiterà la propria query e dopo aver selezionato il tasto **research** gli verranno mostrati tutti i composti organici relativi alla query digitata, tramite il tasto **Get Query** l'utente andrà a scaricare tutti i file dei composti organici in formato *.pdb*. Nel caso di esempio sono state scelte tutte le proteine dei recettori dell'Apis Mellifera come mostrato nelle foto: 3.2 e 3.3, per un totale di 60 file *.pdb* contenenti le strutture di determinate proteine. La lista delle proteine scaricate è disponibile nell'appendice: B.1.

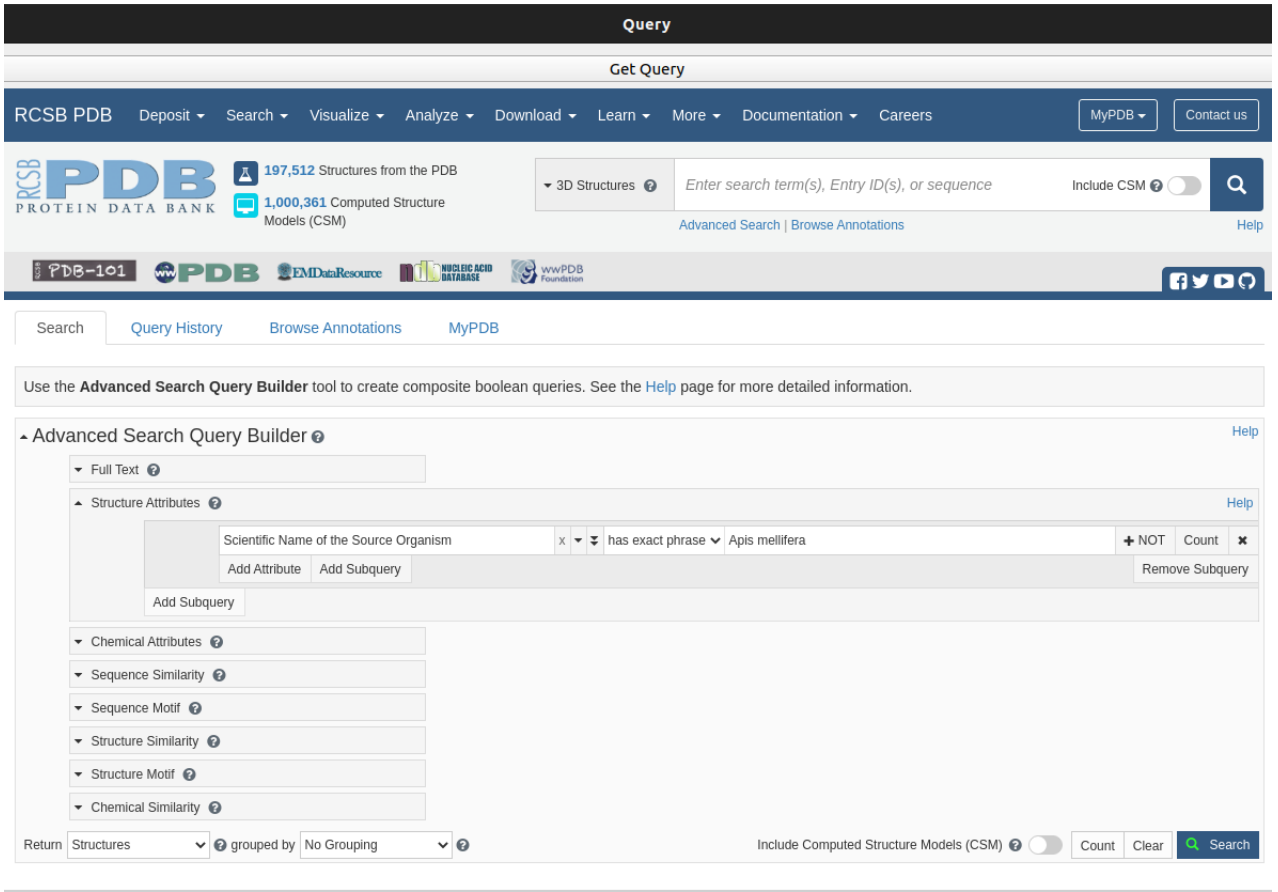


Figura (3.2): Query dei recettori

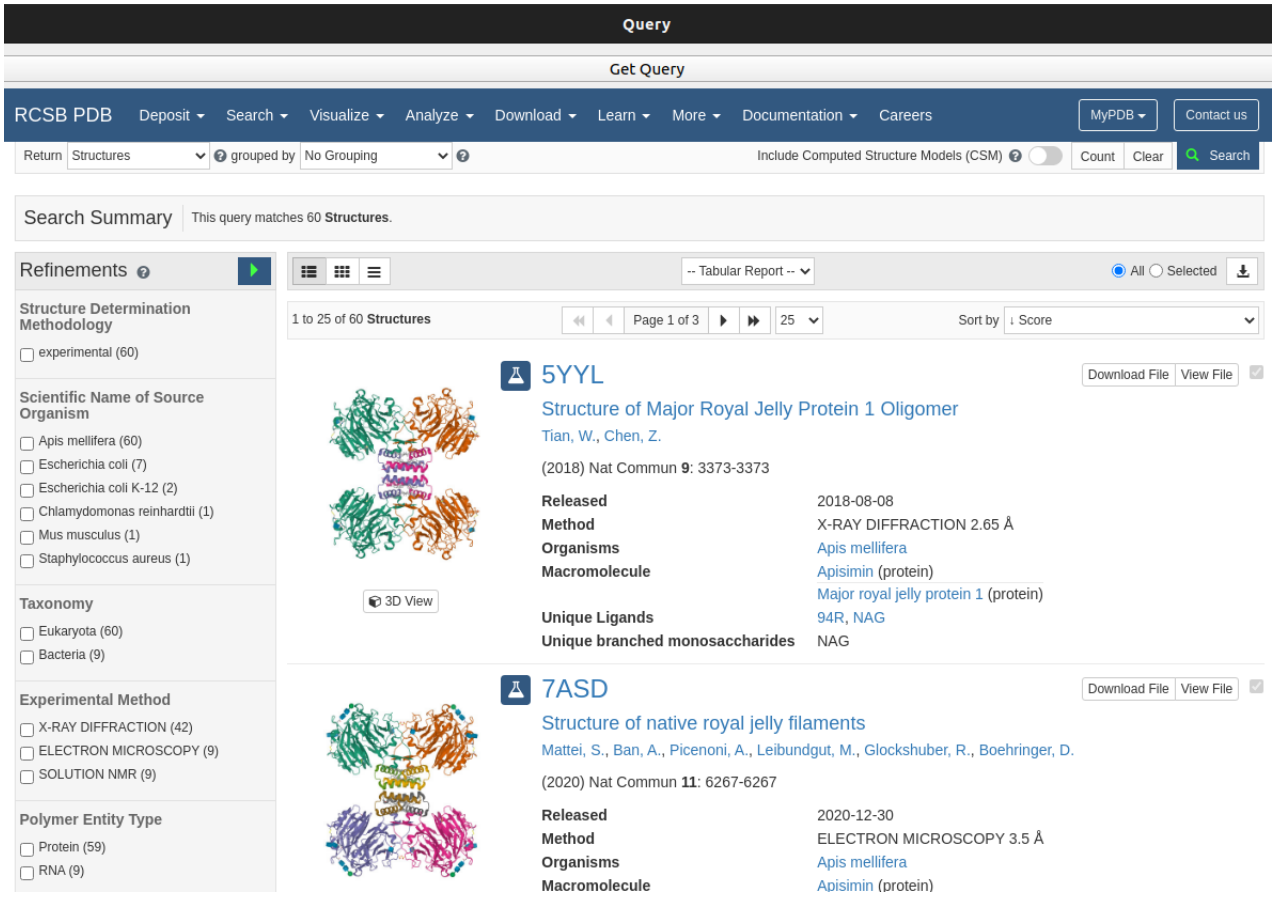


Figura (3.3): Proteine dei recettori dell'apis mellifera

3.4 Preparazione dei ligandi e dei recettori

Il primo step propedeutico per il docking è la preparazione dei ligandi e dei recettori, questa fase viene esplicitamente eseguita dal software realizzato. L'intera fase è svolta: per i ligandi dallo script python **prepare_ligands.py** mentre, per i recettori dallo script python **prepare_receptors.py**. Se l'applicazione viene utilizzata mediante GUI la preparazione dei ligandi e dei recettori può essere effettuata mediante la sezione **Preparation** della pagina principale come mostrato in

figura 3.4.

- Selezionando l'opzione **Ligands** si accede alla sezione relativa alla preparazione dei ligandi
- Selezionando l'opzione **Receptors** si accede alla sezione relativa alla preparazione dei recettori
- Selezionando l'opzione **Back** si ritorna alla pagina principale.

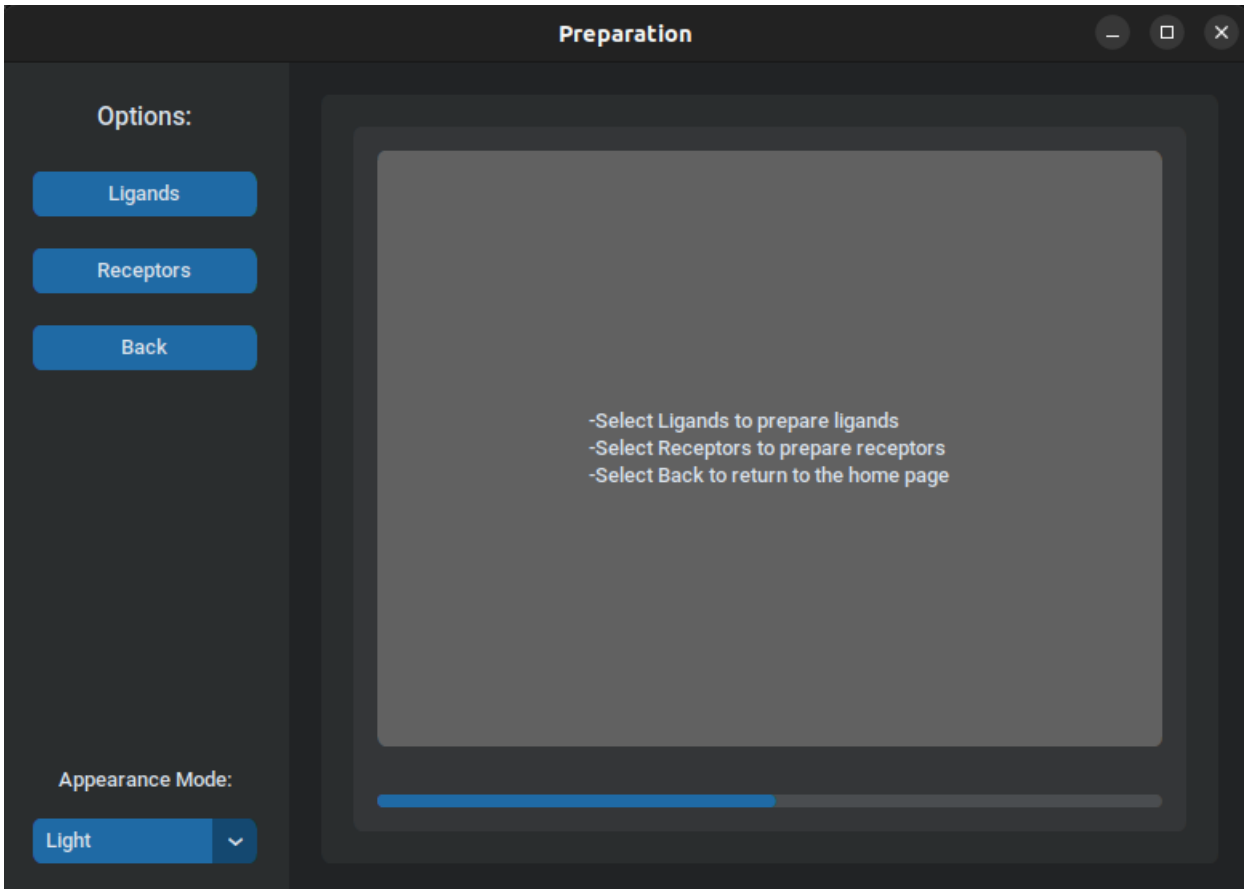


Figura (3.4): Sezione **Preparation** della GUI

3.4.1 Preparazione dei ligandi tramite script

Lo script python **prepare_ligands.py**, che esegue tale fase, viene eseguito da terminale mediante il comando:

```
python prepare_ligands.py
```

Questo script può ricevere diversi argomenti in input:

- $[-v]|[--verbose]$: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- $[-e]|[--input-file]$: specifica un nuovo file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi, se non viene specificata tale opzione verrà scelto il file di default scaricato insieme al software
- $[-E]|[--excel-folder]$: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-s]|[--sdf-folder]$: specifica la directory dei file *.sdf* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-P]|[--pdb-folder]$: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-p]|[--pdbqt-folder]$: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default

- $[-k]|[--keep-ligands]$: specificando questa opzione viene scelto di non cancellare i file dei ligandi precedentemente scaricati, se non viene specificata tale opzione i file verranno cancellati
- $[-h]|[--help]$: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, e nel caso siano stati inseriti in input verrà controllata l'esistenza e la validità degli stessi. Nel caso non siano presenti le directory contenenti i file *.sdf*, *.pdb* e *.pdbqt*, queste verranno create automaticamente dall'applicazione a meno di input inseriti dall'utente e saranno cancellati i file precedentemente scaricati a meno di input contrario.
Lo script richiama la funzione **selectLigands** la quale si occupa di scaricare da **Pubchem** i file *.sdf* corrispondenti alla lista dei ligandi in input. La funzione prende in input:

- *input_path*, il path del file di input
- *sdf_folder*, il path della directory di output per i file *.sdf*
- *excel_folder*, il path della directory di output per i file *.xlsx*
- *verbose*, il flag relativo al verbose.

In output ci vengono dati i file *.sdf* corrispondenti ai ligandi in input, dove il nome di ogni file è preceduto dal suffisso *ligand_*, e un file *.xlsx* contenente i risultati dell'operazione di download nominato *ligands_sdf_output.xls*.

```
selectLigands(input_path, sdf_folder, excel_folder, verbose)
```

L'interfaccia con il database avviene tramite il pacchetto **PubChemPy**, in particolare la funzione **pubchempy.get_compounds** permette di ricercare nel database il record relativo allo specifico ligando il cui nome, a cui viene precedentemente aggiunto il suffisso *ligand_*, gli viene fornito in input. Questa prende in input:

- *identifier*, il composto da ricercare nel database
- *namespace*, il parametro in base al quale ricercare il composto, nel nostro caso il parametro scelto è il nome indicato dalla stringa *"name"*
- *record_type* ovvero il tipo di record relativo al composto in questione da scaricare, nel nostro caso il parametro scelto è la stringa *"3d"* che indica la struttura 3D del ligando.

In output verrà fornito il record della struttura 3D del ligando in questione.

```
pubchempy.get_compounds(identifier, "name", record_type="3d")
```

La funzione che effettivamente si occupa di scaricare la struttura 3D del ligando in formato *.sdf*, richiamata da **selectLigands** è **pubchempy.download**, prende in input:

- *outformat*, il formato in cui deve essere scaricato il file, nel nostro caso il parametro scelto è il formato *.sdf* indicato dalla stringa *"SDF"*
- *path*, il path della directory in cui vengono scaricati i file *.sdf*

- *identifier*, il composto da scaricare nel database
- *namespace*, il parametro in base al quale ricercare il composto, nel nostro caso il parametro scelto è il nome indicato dalla stringa *"name"*
- *record_type* ovvero il tipo di record relativo al composto in questione da scaricare, nel nostro caso il parametro scelto è la stringa *"3d"* che indica la struttura 3D del ligando.

L'output di tale funzione sarà il file *.sdf* del ligando in questione.

```
pubchempy.download("SDF", path, identifier, "name", record_type="3d")
```

I nomi di alcuni file potrebbero non essere presenti nel database oppure, a causa di spazi presenti nel loro nome, non essere riconosciuti. Nel primo caso i ligandi vengono semplicemente scartati nel secondo caso gli spazi vengono sostituiti da underscore (_), seguendo quindi la nomenclatura del database, e viene effettuata nuovamente la ricerca di questi nel database. Se anche stavolta la ricerca non ha successo il ligando in questione viene definitivamente scartato.
Oltre ai file *.sdf* la **selectLigands** produrrà anche un file *.xlsx* contenente il riepigolo dei risultati ottenuti dalla funzione, ovvero:

- file scaricati
- file scaricati con il nome modificato
- ligandi non trovati all'interno del database.

Nel caso di esempio, dei 297 ligandi in input, 289 sono stati trovati e scaricati con successo, i restanti 8 sono stati scartati: 3.1.

Sodium 4-nitrophenolate	Silthiofa
Sodium 2-methoxy-5-nitrophenolate	Potassium bicarbonate
(E,E)-7,9-Dodecadien-1-yl acetate	Dodine
Sodium 2-nitrophenolate	Ziram

Tabella (3.1): Ligandi non scaricati

Dopo aver scaricato i file *.sdf* questi devono essere convertiti in formato *.pdb*, per fare ciò lo script richiama la funzione **sdf2pdb**, la quale prende in input:

- *sdf_folder*, la directory con i file *.sdf* in input
- *pdb_folder*, la directory con i file *.pdb* in output
- *verbose*, il flag relativo al verbose.

La funzione restituisce in output i file *.pdb* corrispondenti a tutti i file *.sdf* dati in input.

```
sdf2pdb(sdf_folder, pdb_folder, verbose)
```

La funzione usufruisce del programma **Open babel** in particolare esegue la conversione da *.sdf* a *.pdb* richiamando la sua versione da terminale tramite il comando **obabel**:

```
obabel sdf_file_path -O pdb_path
```

Nell'istruzione sopra *sdf_file_path* indica la directory con i file *.sdf* in input, la directory con i file *.pdb* in output, *pdb_path*, viene specificata tramite lo switch *-O*. Tramite tale istruzione è possibile effettuare la conversione di un singolo file, la

conversione di tutti i *.sdf* avviene ciclando su tutti i file nella directory corrispondente.
L'ultimo step nella preparazione dei ligandi è effettuato dalla funzione **prepareLigands**, questa funzione prende in input:

- *pdb_folder*, la directory con i file *.pdb* in input
- *pdbqt_folder*, la directory con i file *.pdbqt* in output
- *verbose*, il flag relativo al verbose.

La funzione restituisce la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

```
prepareLigands(pdb_folder, pdbqt_folder, verbose)
```

prepareLigands usufruisce dello script **prepare_ligand4** offerto dalla suite **ADFR** tramite il quale effettua la corretta conversione da *.pdb* a *.pdbqt*. Lo script viene richiamato tramite riga di comando dalla funzione mediante il comando **prepare_ligand**:

```
prepare_ligand -l pdb_file_path -v -o pdbqt_path
```

Nell'istruzione sopra, *pdb_file_path* indica la directory con i file *.pdb* in input ed è specificata tramite lo switch *-l*, lo switch *-v* indica che è attivato il verbose e la directory con i file *.pdbqt* in output, *pdbqt_path*, viene specificata tramite lo switch *-o*. Tramite tale istruzione è possibile effettuare la conversione di un singolo file, la conversione di tutti i *.pdb* avviene ciclando su tutti i file nella directory corrispondente.

3.4.2 Preparazione dei ligandi tramite GUI

La preparazione dei ligandi tramite GUI avviene selezionando il tasto **Ligands** nella sezione **Preparation** del software. All'inteno della pagina relativa alla preparazione dei ligandi premendo il tasto **Execute** è possibile effettuare i procedimenti precedentemente spiegati nella sezione relativa all'esecuzione tramite script (3.4.1). Premendo il tasto **Back** è possibile tornare alla sezione **Preparation** relativa alla preparazione degli input. Come si osserva nella figura 3.5 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- un file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi
- la directory dei file *.xlsx* di output
- la directory dei file *.sdf* di output
- la directory dei file *.pdb* di output
- la directory dei file *.pdbqt* di output.

Se lasciate vuote queste entry verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale se selezionato permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.

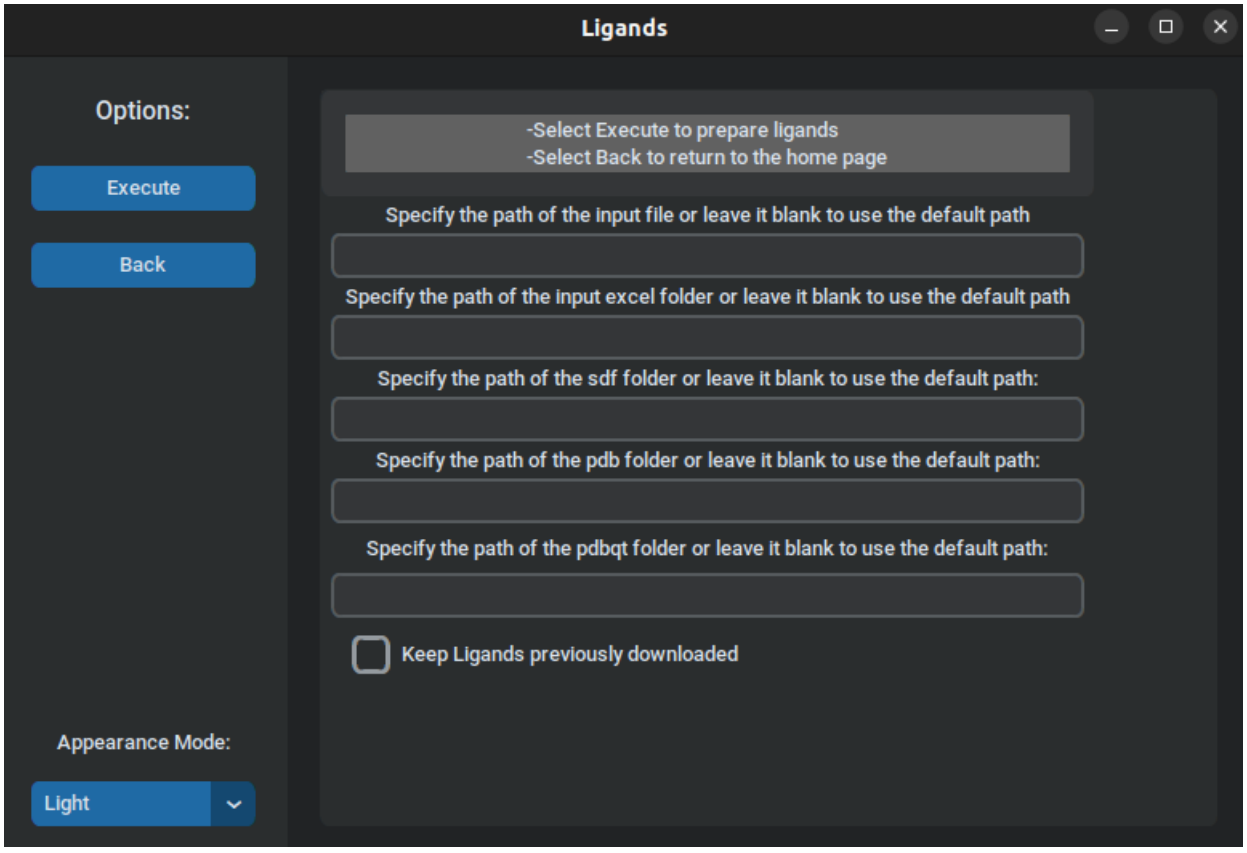


Figura (3.5): Sezione **Ligands** della GUI

La GUI per la preparazione dei ligandi richiama versioni modificate delle funzioni e dello script **prepare_ligands.py** discussi nella precedente sezioni, le modifiche apportate semplicemente adattano le procedure e gli script all'interfaccia grafica senza eliminare alcuna funzionalità precedentemente illustrata per la versione da riga di comando.
Durante l'esecuzione vengono mostrate diverse **progress bar** (figura: 3.6, 3.7 e 3.8) in modo tale che sia restituito un feedback all'utente relativo al progresso della preparazione dei ligandi. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.9) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.10).

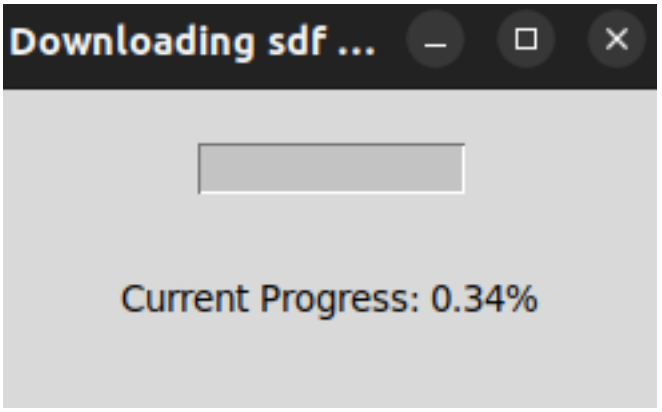


Figura (3.6): **Progress bar** dei file *.sdf*

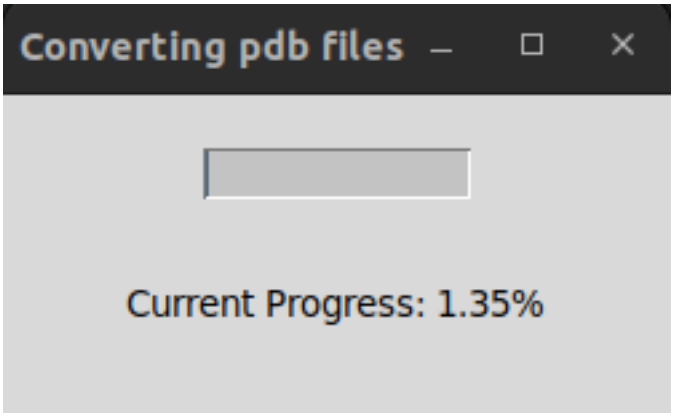


Figura (3.7): **Progress bar** dei file *.pdb*

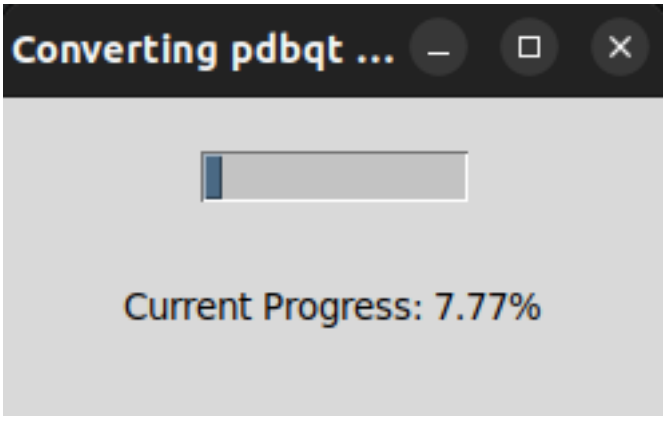


Figura (3.8): **Progress bar** dei file *.pdbqt*

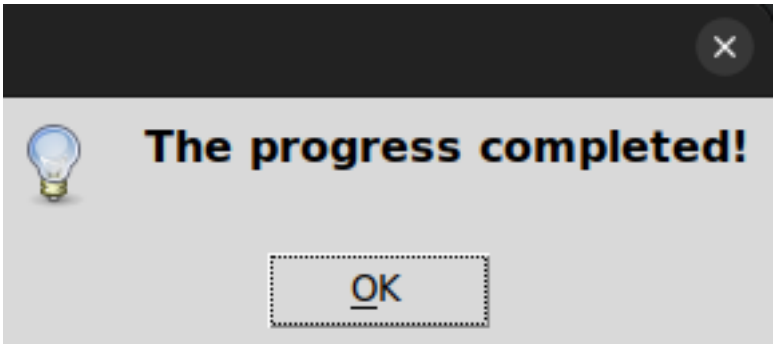


Figura (3.9): Messaggio processo completato con successo

Figura (3.10): Messaggio di errore

3.4.3 Preparazione dei recettori tramite script

o script python **prepare_receptors.py**, che esegue tale fase, viene eseguito da terminale mediante il comando:

```
python prepare_receptors.py
```

Questo script può ricevere diversi argomenti in input:

- $[-v]|[--verbose]$: serve per attivare il verbose e se non viene specificata tale opzione viene lasciata di default inattivo
- $[-E]|[--excel-folder]$: specifica la directory dei file *.xlsx* di output, se non viene specificata tale opzione verrà scelta la directory di default

- $[-P]|[--pdb-folder]$: specifica la directory dei file *.pdb* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-p]|[--pdbqt-folder]$: specifica la directory dei file *.pdbqt* di output, se non viene specificata tale opzione verrà scelta la directory di default
- $[-k]|[--keep-pdb-files]$: specificando questa opzione viene scelto di non cancellare i file delle proteine precedentemente scaricati, se non viene specificata tale opzione i file verranno cancellati
- $[-m]|[--margin]$: specificando questa opzione viene fornita in input la dimensione in *angstrom* del margine della *grid box* per le proteine se non viene specificata tale opzione viene impostato il valore di default ovvero 3
- $[-h]|[--help]$: stampa la spiegazione degli input per lo script.

Lo script quando eseguito andrà ad inizializzare i vari percorsi, e nel caso siano stati inseriti in input verrà controllata l'esistenza e la validità degli stessi. Nel caso non siano presenti le directory contenenti i file *.txt*, *.pdb* e *.pdbqt*, queste verranno create automaticamente dall'applicazione a meno di input inseriti dall'utente e saranno cancellati i file precedentemente scaricati a meno di input contrario.
Lo script richiama la funzione **selectReceptors** la quale si occupa di scaricare da **Pubchem** i file *.pdb* corrispondenti alle proteine scelte. La funzione prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *excel_folder*, il path della directory di output per i file *.xlsx*
- *verbose*, il flag relativo al verbose.

In output ci vengono dati i file *.pdb* corrispondenti ai recettori scelti in input e un file *.xlsx* contenente i risultati dell'operazione di download nominato *info_proteins.xls*.

```
selectReceptors(pdb_folder, excel_folder, verbose)
```

All'interno di questa **selectReceptors** viene richiamata la funzione **RestApiSelection** la quale prende in input l'*URL* di PubChem:

```
RestApiSelection(URL)
```

Questa funzione apre una **web view** sul sito di PubChem come mostrato nella figura 3.2, l'utente potrà digitare la propria query e scaricare i composti scelti selezionando il tasto in alto **Get Query**.
La **web view** restituisce l'**URL** della pagina del database contenente i composti selezionati dalla query, il *JSON* di tale pagina viene estratto e convertito nella struttura dati *dizionario* di python tramite la funzione **loads** della libreria **json**.

```
dictionary = json.loads(data)
```

Nel *dizionario* restituito dalla funzione sarà contenuto l'elenco dei composti selezionati tramite la query.
Il software rieseguirà direttamente la query tramite la funzione **get** contenuta nella libreria **requests**. Questa funzione permette di inviare una richiesta di tipo *HTTP/1.1* prendendo in

input **URL** della query al quale viene aggiunto l'elenco di composti da scegliere in formato *JSON*, ciò eseguito convertendo il *dizionario* precedentemente ottenuto in **JSON** mediante la funzione **dump** della libreria **json**:

```
dictionary = json.dump(dictionary)
```

La funzione **get** restituisce la lista di proteine da scaricare:

```
requests.get(f"https://search.rcsb.org/rcsbsearch/v2/query?json={dictionary}")
```

La funzione **selectReceptors** richiama successivamente la funzione **downloadPdbs** che prende in input:

- *pdb_list*, la lista di proteine precedentemente ottenuta
- *outputh_path*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

La funzione ritorna in output i file *.pdb* delle proteine contenute nella lista in input:

```
downloadPdbs(pdbs_list, output_path, verbose)
```

La funzione **downloadPdbs** richiama la funzione **fetchPDB** del pacchetto **ProDy**, questa prende in input:

- *protein_code*, la proteina da scaricare
- *folder*, il path della directory di output per i file *.pdb*, nel nostro caso sarà impostato ad *out_path*

- *compressed*, il flag per decidere se il file scaricato dovrà essere compresso oppure no, nel nostro caso il flag sarà impostato a *False* per indicare che i file scaricati debbano essere già decompressi.

Tramite tale istruzione è possibile interfacciarsi con **PubChem** e scaricare di un singolo file *.pdb* mediante richiesta *FTP*, il download di tutte le proteine della lista avviene ciclando su tutti gli elementi di essa:

```
fetchPDB(protein_code, folder=output_path, compressed=False)
```

Dopo aver effettuato il download di tutte i file *.pdb* le informazioni relative al download ed ai file scaricati vengono salvati in un file *.xlsx* chiamato *info_proteins.xlsx* salvato nella directory */output/*.
Lo script **prepare_receptors.py** una volta terminata la funzione **selectReceptors** richiama **splitRepeatedResidues** che prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di rimuovere i residui ripetuti memorizzati nel file *.pdb* sotto la dicitura *alternative location* o *alt_loc*:

```
splitRepeatedResidues(pdb_folder, verbose, output_folder=None)
```

Per accedere agli attributi dei file *.pdb* utilizziamo la funzione **read_pdb** del pacchetto **PandasPdb**, questa prende in input il nome del file *.pdb*:

```
PandasPdb.read_pdb(pdb_file)
```

In particolare gli attributi *A* e *B* sono quelli ripetuti e i *B* saranno quelli eliminati nella colonna *alt_loc*.
Successivamente viene richiamata dallo script la funzione **deleteHeteroatomsChains**, la quale prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di rimuovere le catene di eteroatomi contenute nel file *.pdb*, queste sono evidenziate nel file tramite un loro attributo ovvero la keyword *HETATM*, una volta rimosse, le catene vengono salvate.

```
deleteHeteroatomsChains(pdb_folder, verbose)
```

L'estrazione degli attributi dai file *.pdb* avviene anche in questo caso tramite la funzione **read_pdb** precedentemente descritta.
prepare_receptors.py richiama successivamente la funzione **splitChains** che prende in input:

- *pdb_folder*, il path della directory di output per i file *.pdb*
- *verbose*, il flag relativo al verbose.

Questa funzione si occupa di dividere le catene di residui che formano la struttura della proteina mantenendo soltanto le catene distinte, cioè quelle catene che hanno solo una sequenza aminoacidica distinta e non ripetuta da altre catene.

```
splitChains(pdb_folder, verbose)
```

Tramite la funzione **parsePDB** della libreria **prody** viene ricostruita la struttura proteica attraverso oggetti come: la lista delle molecole, la lista degli atomi e dei residui, questa viene rappresentata da un oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**.

```
atoms = parsePDB(protein_file)
```

La ricostruzione della struttura proteica avviene tramite la vista gerarchica della struttura, ciò viene implementata tramite il metodo **getHierView** della classe **AtomGroup**:

```
atoms.getHierVieW()
```

Vengono quindi esaminate tutte le sequenze, sempre tramite la funzione **parsePDB**, e scelte solo le sequenze distinte, infine vengono salvate in un file tutte le catene distinte come se fossero delle proteine attraverso la funzione **writePDB** di **prody**, questa funzione prende in input:

- *filename*, il nome del file
- *new_atoms*, il composto da salvare

Questi file prenderanno il nome della proteina principale al quale viene aggiunto un underscore (_) seguito dal nome della catena distinta salvata nel file.

```
writePDB(filename, new_atoms)
```

Lo step successivo nella preparazione dei recettori è effettuato dalla funzione **prepareReceptors**, questa funzione prende in input:

- *pdb_folder*, la directory con i file *.pdb* in input
- *pdbqt_folder*, la directory con i file *.pdbqt* in output
- *verbose*, il flag relativo al verbose
- *charges_to_add*, la carica da aggiungere alle proteine.

La funzione restituisce la conversione in file *.pdbqt* dei corrispondenti i file *.pdb* dati in input.

```
prepareReceptors(pdb_folder, pdbqt_folder, verbose)
```

prepareReceptors utilizza lo script **prepare_receptors4** offerto dalla suite **ADFR**. In realtà viene utilizzata una versione modificata di tale script: **replacePrepareReceptor4.sh**. Questa versione modificata tramite bash scripting sostituisce la carica di Gasteiger con la carica in input ovvero *charges_to_add* che nel nostro caso sono cariche di Kollman come specificate dalla stringa *"Kollman"*. Tramite questo script si effettua la corretta conversione da *.pdb* a *.pdbqt*. Lo script viene richiamato tramite riga di comando dalla funzione mediante il comando **prepare_receptor**:

```
prepare_receptor
-r
pdb_file_path
-A
checkhydrogens
-C
charges_to_add
-e
-o
pdbqt_folder
```

Nell'istruzione sopra:

- *pdb_file_path* indica la directory con i file *.pdb* in input ed è specificata tramite lo switch *-r*
- *checkydrogens* indica l'opzione tramite la quale vengono controllati gli atomi di idrogeno specificata lo switch *-A*
- *charges_to_add* indica la carica usata in input impostata tramite lo switch *-C*
- *pdbqt_folder* indica la directory con i file *.pdbqt* in output specificata tramite lo switch *-o*

Tramite tale istruzione è possibile effettuare la conversione di un singolo file, mentre la conversione di tutti i *.pdb* avviene ciclando su tutti i file nella directory corrispondente.
L'ultima fase nella preparazione dei ligandi consiste nella creazione delle *grid box*, mediante la funzione **createGridboxes** richiamata dallo script **prepare_receptors.py**, la funzione prende in input:

- **pdb_folder**, la directory contenente i file *.pdb*
- **gridbox_outputh_folder**, la directory di output per le *grid box*
- **margin**, la dimensione in *angstrom* del margine della *grid box*
- **verbose**, il flag relativo al verbose.

Per ogni proteina ottenuta deve essere creata una *grid box*, questa definisce lo spazio conformazionale dove si colloca la proteina, è definita da:

- Un centro
- Le dimensioni
- L'esaustività, la quale sarà utilizzata da qualsiasi software da **AutoDock Vina** per la ricerca delle pose possibili del ligando.

Una *grid box* definisce la regione delle proteine dove il docking viene effettuato. Qualsiasi altra regione al di fuori dalla *grid box* non verrà presa in considerazione durante il processo di docking.
Le *grid box* sono un input richiesto da **AutoDock Vina** ma non da altri software per il docking.

```
createGridboxes(pdb_folder, gridbox_output_folder, margin, verbose)
```

Per ogni proteina vengono estratti gli attributi contenuti nel proprio file e viene richiamata da **createGridboxes** la funzione **createGridbox**, questa prende in input:

- **ppdb**, l'insieme di attributi del file *.pdb* ottenuti come oggetto della classe **AtomGroup** restituito dalla funzione **parsePDB**
- **protein_path**, il file *.pdb* corrispondente ad una singola proteina
- **gridbox_outputh_folder**, la directory di output per le *grid box*
- **margin**, la dimensione in *angstrom* del margine della *grid box*

- **verbose**, il flag relativo al verbose.

La funzione **createGridbox** costruisce una *grid box* per una singola proteina utilizzando gli attributi del file *.pdb* rispettando i parametri precedentemente definiti per una *grid box*.

3.4.4 Preparazione dei reccettori tramite GUI

La preparazione dei recettori tramite GUI avviene selezionando il tasto **Receptors** nella sezione **Preparation** del software. All'inteno della pagina relativa alla preparazione dei recettori premendo il tasto **Execute** è possibile effettuare i procedimenti precedentemente spiegati nella sezione relativa all'esecuzione tramite script (3.4.3). Premendo il tasto **Back** è possibile tornare alla sezione **Preparation** relativa alla preparazione degli input. Come si osserva nella figura 3.11 all'interno di tale sezione sono presenti delle **entry** dove è possibile specificare diversi input tra cui:

- un file di input (*.xlsx*, *.xls*) o (*.txt*) da cui prendere i nomi dei ligandi
- la directory dei file *.xlsx* di output
- la directory delle *grid box* di output *.txt*
- la directory dei file *.pdb* di output
- la directory dei file *.pdbqt* di output
- il valore del margine delle *grid box*
- la carica da aggiungere.

Se lasciate vuote queste entry verrano impostate le configurazioni di default. E' presente anche un **checkbox** il quale se selezionato permette di mantenere i file precedentemente scaricati che altrimenti verrebbero cancellati.

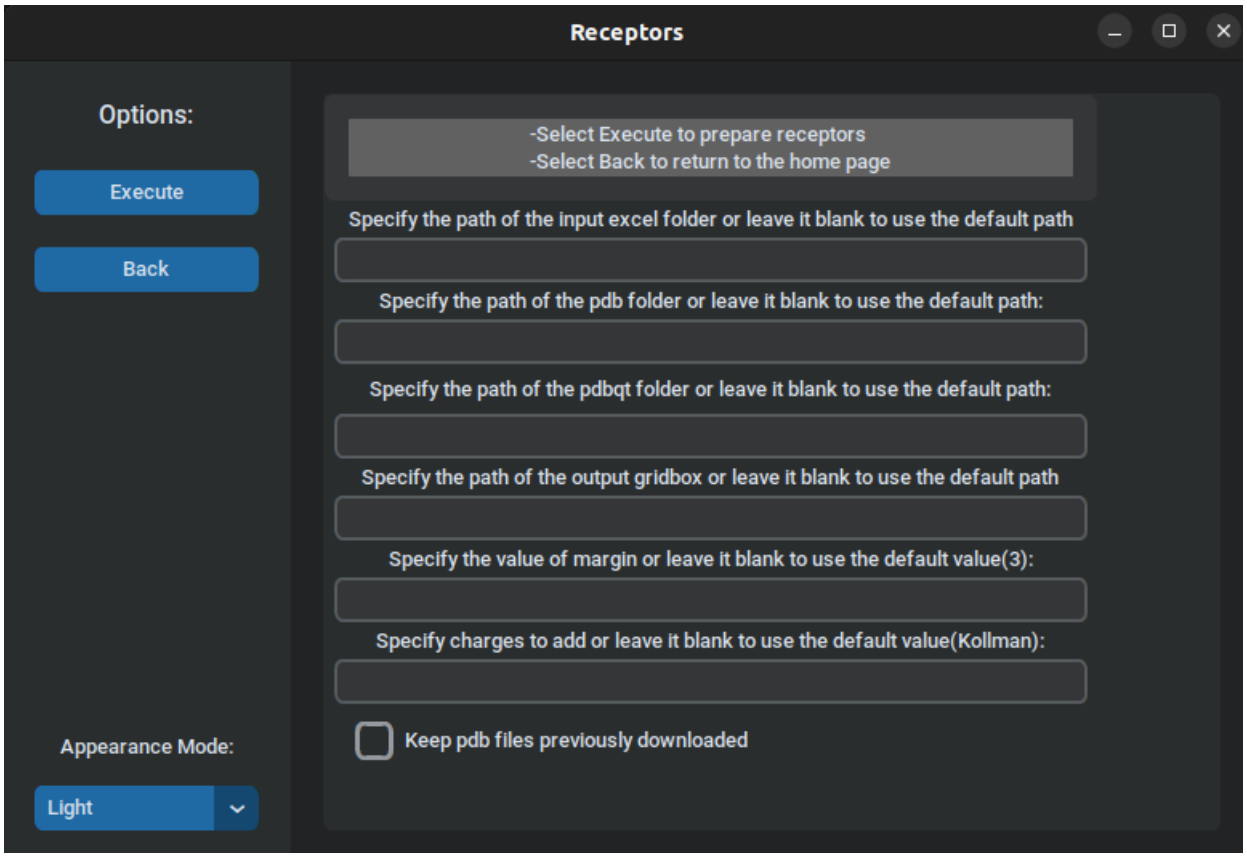


Figura (3.11): Sezione **Receptors** della GUI

La GUI per la preparazione dei recettori richiama versioni modificate delle funzioni e dello script **prepare_receptors.py** discussi nella precedente sezioni, le modifiche apportate semplicemente adattano le procedure e gli script all'interfaccia grafica senza eliminare alcuna funzionalità precedentemente illustrata per la versione da riga di comando.
Durante l'esecuzione viene mostrata una **progress bar** (figu-

ra: 3.12) in modo tale che sia restituito un feedback all'utente relativo al progresso della preparazione dei ligandi. Al completamento di ogni fase sarà restituito un messaggio di avviso (figura: 3.13) ed in caso di input non valido sarà restituito un messaggio di errore (figura: 3.14).

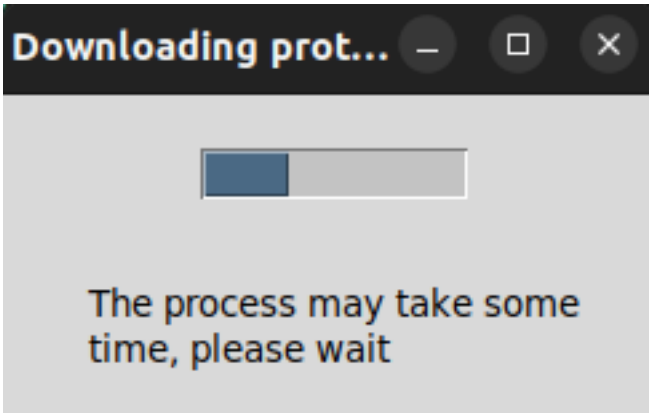


Figura (3.12): **Progress bar** delle proteine

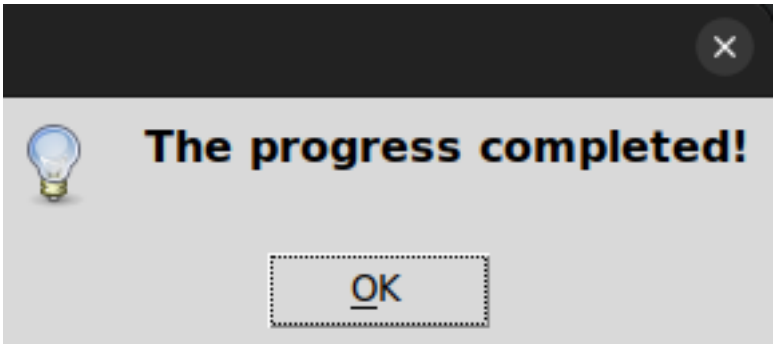


Figura (3.13): Messaggio processo completato con successo

Figura (3.14): Messaggio di errore

3.5 Docking

Capitolo 4

Conclusioni

Appendice A

Tabella dei ligandi

(E,E)-7,9-Dodecadien-1-yl acetate	Gibberellins
(E,E)-8,10-Dodecadien-1-ol	Glyphosate
(3E,8Z,11Z)-Tetradeca-3,8,11-trienyl acetate	Halauxifen-methyl
3E,8Z-Tetradecadienyl acetate	Halosulfuron-methyl
(E)-11-Tetradecen-1-yl acetate	Hexythiazox
(E)-5-Decen-1-ol	Hymexazol
(E)-5-Decen-1-yl acetate	Imazalil
(E)-8-Dodecen-1-yl acetate	Imazamox
(Z,E)-7,11-Hexadecadien-1-yl acetate	Indolylbutyric acid
(Z,E)-Tetradeca-9,11-dienyl acetate	Indoxacarb
(Z,E)-9,12-Tetradecadien-1-yl acetate	Iodosulfuron
(Z)-11-Hexadecen-1-ol	Ipconazole
(Z)-11-Hexadecen-1-yl acetate	Iprovalicarb
(Z)-11-Hexadecenal	Isofetamid
(Z)-11-Tetradecen-1-yl acetate	Isopyrazam
(Z)-13-Octadecenal	Isoxaben
(Z)-7-Tetradecenal	Isoxaflutole
(Z)-8-Dodecen-1-ol	Kresoxim-methyl
(Z)-8-Dodecen-1-yl acetate	L-Ascorbic acid
(Z)-8-Tetradecen-1-ol	lambda-Cyhalothrin
z-8-Tetradecenyl acetate	Laminarin
(Z)-9-Dodecen-1-yl acetate	Lauric acid
(Z)-9-Hexadecenal	Lavandulyl senecioate
(Z)-9-Tetradecen-1-yl acetate	Lenacil
1-Decanol	Malathion
1-methylcyclopropene	Maleic hydrazide
1-Naphthylacetamide	Mandestrobin
1-Naphthylacetic acid	Mandipropamid

Continua nella pagina successiva

1,4-Dimethylnaphthalene	MCPA
2-Phenylphenol	MCPB
2,4-D	Mecoprop-P
Methyl 2,5-dichlorobenzoate	Mefentrifluconazole
24-Epibrassinolide	Mepanipyrim
4-(2,4-Dichlorophenoxy)butanoic acid	Mepiquat
6-Benzyladenine	Meptyldinocap
8-Hydroxyquinoline	Mesosulfuron
Acequinocyl	Mesotrione
Acetamiprid	Metaflumizone
Acetic acid	Metalaxyl
Acibenzolar-S-methyl	Metalaxyl-M
Aclonifen	Metaldehyde
Acrinathrin	Metam
Ametoctradin	Metamitron
Amidosulfuron	Metazachlor
Aminopyralid	Metconazole
Amisulbrom	Methoxyfenozide
Azimsulfuron	Methyl decanoate
Azoxystrobin	Methyl octanoate
Beflubutamid	Zineb
Benalaxyl-M	Metobromuron
Benfluralin	Metrafenone
Bensulfuron	Metribuzin
Bentazone	Metsulfuron-methyl
Benthiavalicarb	Milbemectin
Benzoic acid	Tetradecyl acetate
Benzovindiflupyr	Napropamide
Bifenazate	Nicosulfuron
Bifenox	Oleic acid
Bispyribac	Orange oil
Bixafen	Oxamyl
Boscalid	Oxathiapiprolin
Bromuconazole	Oxyfluorfen
Bupirimate	Paclobutrazol
Buprofezin	Pelargonic acid
Capric acid	Penconazole
Caprylic acid	Pendimethalin
Captan	Penflufen
Carfentrazone-ethyl	Penoxsulam
Carvone	Penthiopyrad
Chlorantraniliprole	Pethoxamid
	Continua nella pagina successiva

Chlormequat	Phenmedipham
Chlorotoluron	Phosmet
Chromafenozide	Phosphane
Clethodim	Picloram
Clodinafop	Picolinafen
Clofentezine	Pinoxaden
Clomazone	Pirimicarb
Clopyralid	Pirimiphos-methyl
Cyantraniliprole	Potassium bicarbonate
Cyazofamid	Prochloraz
Cycloxydim	Prohexadione
Cyflufenamid	Propamocarb
Cyflumetofen	Propaquizafop
Cyhalofop-butyl	Propoxycarbazone
Cymoxanil	Propyzamide
Cypermethrin	Proquinazid
Cyprodinil	Prosulfocarb
Daminozide	Prosulfuron
Dazomet	Prothioconazole
Deltamethrin	Pyraclostrobin
Dicamba	Pyraflufen-ethyl
Dichlorprop-P	Pyridaben
Diclofop	Pyridalyl
Difenoconazole	Pyridate
Diflufenican	Pyrimethanil
Dimethachlor	Pyriofenone
Dimethenamid-P	Pyriproxyfen
Dimethomorph	Pyroxsulam
Dimoxystrobin	Quinmerac
Dithianon	Quizalofop-P
Dodecan-1-ol	Quizalofop-P-ethyl
Dodecyl acetate	Quizalofop-P-tefuryl
Dodemorph	Rescalure
Dodine	Rimsulfuron
Ethephon	S-Metolachlor
Ethofumesate	Sedaxane
Etofenprox	Sodium 2-methoxy-5-nitrophenolate
Etoxazole	Sodium 2-nitrophenolate
Eugenol	Sodium 4-nitrophenolate
Fenazaquin	Spiromesifen
Fenhexamid	Spirotetramat
Fenoxaprop-P	Spiroxamine
Continua nella pagina successiva	

Fenpicoxamid	Sulcotrione
Fenpropidin	Sulfosulfuron
Fenpyrazamine	Sulfoxaflor
Fenpyroximate	Sulfuryl fluoride
Flazasulfuron	tau-Fluvalinate
Flonicamid	Tebuconazole
Florasulam	Tebufenozide
Florpyrauxifen-benzyl	Tebufenpyrad
Fluazifop-P	Tefluthrin
Fluazinam	Tembotrione
Flubendiamide	Terbuthylazine
Fludioxonil	Tetraconazole
Flufenacet	Tetradecan-1-ol
Flumetralin	Thiabendazole
Flumioxazin	Thiencarbazone-methyl
Fluometuron	Thifensulfuron-methyl
Fluopicolide	Thymol
Fluopyram	Tolclofos-methyl
Fluoxastrobin	Tri-allate
Flupyradifurone	Tribenuron
Fluquinconazole	Triclopyr
Flurochloridone	Trifloxystrobin
Fluroxypyr	Triflusulfuron
Flutianil	Trinexapac
Flutolanil	Triticonazole
Fluxapyroxad	Tritosulfuron
Folpet	Urea
Foramsulfuron	Valifenalate
Forchlorfenuron	Ziram
Formetanate	Zoxamide
Fosetyl	Quartz sand
Fosthiazate	Silthiofam
Gamma-cyhalothrin	Esfenvalerate
Garlic extract	Ethylene
Geraniol	Gibberellic acid
(2Z,4E)-5-[(1S)-1-Hydroxy-2,6,6-trimethyl-4-oxocyclohex-2-en-1-yl]-3-methylpenta-2,4-dienoic acid	
1-(4-Chlorophenyl)-5-(2-methoxyethoxy)-4-oxo-1,4-dihydrocinnoline-3-carboxylic acid	

Tabella (A.1): Tabella dei ligandi in input

Appendice B

Tabella dei recettori

5YYL	3BFA	3D78	6LQK	2MLT
7ASD	3BFB	3FE6	6O4M	2MW6
1BH1	3BFH	3FE8	7OXF	2N8V
1CCV	3BJH	3FE9	7ZS6	3D75
1FCQ	3CAB	3R72	2J88	3D76
1FCU	3CDN	3RZS	3QRX	3D77
1FCV	3CYZ	3S0A	6GXN	5OHX
1POC	3CZ0	3S0B	6GXP	5XZ3
1TER	3CZ1	3S0D	4E81	6DST
1TUJ	3CZ2	3S0E	6GXO	6YSS
2H8V	3D73	3S0F	6GWT	6YSU
2LIC	3D74	3S0G	6YST	5O2R

Tabella (B.1): Tabella dei recettori in input

Bibliografia

[1] V Bellucci, P Bianco e V Silli. *Le api, sentinelle dell'inquinamento ambientale*.

[2] Marianna Martinello et al. ≪Spring mortality in honey bees in northeastern Italy: detection of pesticides and viruses in dead honey bees and other matrices≫. In: *Journal of Apicultural Research* 56.3 (2017), pp. 239–254.

[3] Xuan-Yu Meng et al. ≪Molecular docking: a powerful approach for structure-based drug discovery≫. In: *Current computer-aided drug design* 7.2 (2011), pp. 146–157.

[4] Isabella A Guedes, Camila S de Magalhães e Laurent E Dardenne. ≪Receptor–ligand molecular docking≫. In: *Biophysical reviews* 6.1 (2014), pp. 75–87.

[5] Nataraj S Pagadala, Khajamohiddin Syed e Jack Tuszynski. ≪Software for molecular docking: a review≫. In: *Biophysical reviews* 9.2 (2017), pp. 91–102.

[6] Jiyu Fan, Ailing Fu e Le Zhang. ≪Progress in molecular docking≫. In: *Quantitative Biology* 7.2 (2019), pp. 83–89.