Programlama Laboratuvarı II Proje I
Labirent Çözen Robot Projesi

EMRE KAYA
Öğrenci No
210201029

I. ÖZET

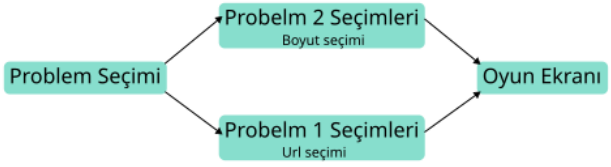
Bu doküman Kocaeli Üniversitesi 20022-2023 bahar dönemi Mühendislik fakültesi Bilgisayar Mühendisliği Programlama Laboratuvarı 2 dersi Proje 1 için hazırlanmıştır. Rapor: Özet, Giriş, Yöntem, Deneysel Sonuçlar ve Katkılar alt başlıklarından oluşmaktadır.

II. GİRİŞ

Proje : Nesneye yönelik programlama ve veri yapıları bilgisi harmanlanarak gerçekleştirilmiştir. Geliştirmede Python dili kullanılmıştır. Problem çözümlerinde herhangi yardımcı bir kütüphane **kullanılmamıştır**. Arayüz geliştirilmesi ve görselliğin sağlanması için **PyGtk** Kullanılmıştır. Ana ekran PyGtk alt sınıfı **Window** miras alınarak gerçekleştirilmişken. Oyun ekranı bu Window eklentili ana ekranın alt sınıfı olarak esasında sadece bir **VBox** objesidir içerisinde de ekrana çizim yapılmasını sağlayan **Gtk Cairo** widgeti miras alınmıştır. Oyun ekranı öncesinde kullanıcıdan verileri adım adım almak amacı ile PyGtk nın sağladığı bir widget olan **stack** yapısı kullanılmıştır.

III. YÖNTEM

- Arayüz : Projede beklenen, Problem 1 ve Problem 2 arasında seçim yapma. Problem 1 için : Url Girişi entery alanı, Url 1 ve Url 2 için sunucudan ön tanımlı dosyaların okunması için kısa yol butonları, Problem seçimine geri dönme butonu bulunmaktadır. Problem 2 için : Labirent boyurlarını x ve y cinsinden girdi alan birer intager entery alanı, labirent zorluğunu belirlemek amaçlı yerleştirilen **Kolay**, **zor** seçim alanı, Problem seçimine geri dönme butonu bulunmaktadır. tüm bu işlemler için Pygtk nın bir widgeti olan stack yapısı kullanılmıştır. Widget animasyonlu sayfa geçişlerine sahip olan ve akış içerisinde kullanıcıdan veri almayı sağlayan bir sistemdir. Yapılan seçimlere göre akış da yön değiştirmektedir. Projenin tümü için toplamda 4 adet stack sayfası kullanılmıştır.



- Oyun ekranı: Oyun ekranı MainWindow içerisindeki bir GtkBox sınıfını miras almaktadır. Main Window'un çocuğudur.

  – Problem 1 parametreleri: Öncesinde seçilen parametreler sayesinde grid ekranı oluşturulur. urlden okunan grid boyutlarında göre rastgele başlangıç ve bitiş noktaları atanır. ardından gride **3x3** ve **2x2** dönüşümleri, istenilen boyutlara uygun önceden tanımlanmış şekiller arasından rastgele bir biçimde seçilerek gerçekleştirilir. Grid tümüyle oluşturulduğunda son olarak robot sınıfı oluşturulur.

  – Problem 2 parametreleri: Öncesinde seçilen parametreler kullanılarak, labirent boyutları içerisinde labirentin başlangıç ve bitiş noktaları atanmaktadır. Atamalar, dörtgen ızgaranın herhangi çapraz 2 köşesi olarak belirlenmektedir. Ardından grid objesi oluşturulur. Grid tümüyle oluşturulduğunda son olarak robot objesi oluşturulur.

  – Labient çizimi: Problemlere göre farklılık göstermekle beraber grid içerisindeli 1 değerleri duvar 0 değerleri ise yol olarak atanır ve çizilir. bunun yanı sıra :
  - Görülmüş fakat herhangi bir şekilde indexlenmemiş noktalar. **Beyaz** zemin ile temsil edilmiştir.
  - Görülmüş Duvarlar. **Duvar** çizimi ile temsil edilmiştir.
  - Başlanıç ve bitiş noktaları, Robotun **şarj istasasyonu** olarak temsil edilmiştir.
  - Kısır olan noktalar: Robot tarafından tüm çevresi çoktan gezilmiş ve herhangi bir şekilde çıkış noktasına giden yola ulaşmasını sağlamayacak olan noktalar. **Turuncu** zemin ile temsil edilmiştir.
  - Kullanılan noktalar: Robotun o noktaya ulaşmasında aktif rol oynayan noktalar- **Mavi** zemin ile temsil edilmiştir.
  - Gidilen noktalar: Robotun o noktaya gittiği fakat bulunduğu noktaya gelmesinde aktif rol oynamayan noktalar. **Sarı** zemin ile temsil edilmiştir.
  - Robot tarafından görülmemiş olan bütün noktaların üzerinde **Yarı saydam** bir bulut katmanı bulunmaktadır. - Problem 1 için robotun her adımında bitişe olan en kısa yolu göstermek

amacıyla bitişe giden yoldaki noktaların üzerinde **Mavi yarı saydam** bir katman eklenir. - Her iki problem için robot labirenti bitirdiğinde bitişe olan en kısa yolu göstermek amacıyla bitişe giden yoldaki noktaların üzerinde **Kırmızı yarı saydam** bir katman eklenir. - Robotun gittiği yolları çizmek için basit bir algoritma mevcuttur: Robot her noktaya nereden girdiğini ve çıktığını not eder. Bu nota bakılarak girdiği yöne ve çıktığı yöne göre bir karenin yarısı uzunluğunda ve 1/5 i genişliğinde aralıklı noktalar çizilir ve her ikisi birleştiğinde robotun ne yöne gittiğini gösteren bir sistem oluşur. - Robotun atadığı derinlik bilgilerine oyun tarafından ulaşılıp ekrana basılması seçeneği **Show Depths** butonu ile sağlanmıştır.

- Robot : yuvarlak bir robot çizimi ile temsil edilmiştir. Labirente herhangi bir bilgisi olmadan **oyun** tarafından atanan bir başlangıç noktasından başlar. Yapabildiği en temel işlev Çevresindeki -Sağ Sol üst alt- kareleri görmek. Çünkü haritaya bırakıldığında harita hakkında hiçbir bilgisi olmamaktadır. ardından 2. temel işlevi gördüğü ve etrafında olan kareler arasında tercih yaparak bir sonraki kareye ilerlemek. Bu tercih sırasında gözden geçirdiği koşullar şunlardır :
  - gitmek istediği Noktanın **Yol** olması gerekmektedir.
  - Robota yazılan algoritmaların birisi de bir gittiği bir noktaya geri adım atmadıkça dönememesidir. Dolayısıyla robot tercihen **önceden gittiği** bir noktaya Gidemez.
  - Robotun gitmemesi istenen bir diğer nokta da **Kısır** noktalardır. Çünkü zaten robot tarafından çoktan sonuca ulaşamayacağı indexlenmiştir.

  Robot Haraket etmeden önce ilk yaptığı sorgu oyun'a gördüğü noktalar arasında bitişin olup olmadığıdır. Çünkü eğer bitiş noktasını gördüyse herhangi diğer noktaya gitmesine gerek yoktur. 2. işlem çevresindeki noktaların kısır olup olmadığı sorgusudur. 3. işlem ise Rastgele bir şekilde gidilebilecek yollar areasında zar atmaktır. Eğer açık bir yol var ise o yola rasgele bir şekilde ilerler. Fakat eğer herhangi bir şekilde gidilebilecek bir yola çevresinde rastlanmazsa, robot hafızasında sakladığı başladığı noktaya kadar geri dönebilen ağaç yapısı ile herhangi açık bir yol bulana kadar bir adım geri döner. Açık bir yol bulduğundaysa ağaç yapısında oradan dallanarak yoluna devam eder.

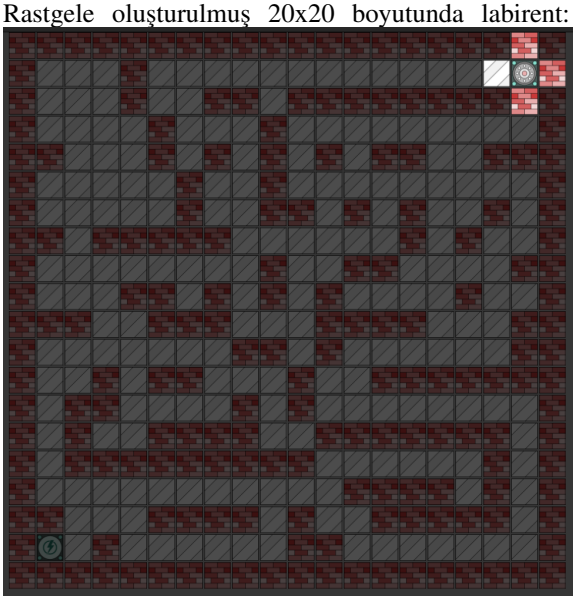
  Robot ayrıca her haraketinde bir noktaya nereden girdiğini ve nereden çıktığını not eder. Bu işlem çizim ve erişim kolaylığı için Noktalarda tutulur. Geri dönme işlevi gerçekleştiğinde bir önceki noktadan çıkma ve aktif bulunulan noktanın girme yönleri sıfırlanır böylece sanki hiç girilmemiş gibi görselleştirilir. Anlaşılması için girilip dönülen noktalar sarı yol olarak indexlenir.

  içerisindeki Algoritmalar :

  - Kısır yol bulma algıritması : Kısır yollar herhangi bir şekilde robotun bildiği diğer yollara çıkan uzantısı bulunmayan nokta veya noktalar topluluğudur. robotun bu yollara girip vakit kaybetmesine gerek yoktur. Tespit işlemi şöyle gerçekleşir: Robot bir noktaya yönelmeden önce o noktanın bildiği kadarıyla kısır olup olmadığını hesaplamaya çalışır. bunun için o noktaya etrafında görülmemiş bir yer olup olmadığını sorar. Eğer var ise bir bilinmezlikten ötürü kesinlikle kısır ataması yapılmaz. Eğer yoksa ve ertafındaki her nokta çoktan kullanılmışsa o nokta kısır atanır. Eğer çevresinde açık yol var ise orasının da kısır olup olmadığı sorulur eğer bu derine inme işlemleri sonucu tüm noktalar kısır ise tüm yol kısır atanır ve girilmeye gerk duyulmaz.

  - Derinlik atama algoritması : Robot için derinlik atama işlemi yalnızca bildiği noktalar ve duvar olmayan Yollar üzerinden gerçekleşir. Robot anlık olarak gördüğü noktaların derinliğini gerekliyse değiştirir. derinlikler ön tanımlı olarak sonsuz olarak ifade edilen -1 değeri ile başlatılır. Labirente robot ilk nırakıldığında derinlik adına yaptığı ilk iş başladığı noktanın derinlik değerini 0 yapmaktır. Ardından çevresindeki noktaları görme işlemini gerçekleştirir ve derinlik atamaya başlar. Gördüğü noktaların derinlikleri -1 değerli ise o noktayı bulunuğu noktadan bir fazla derinlikte atar - Başlangıç çevresi için hepsi 1 derinlik değeri alır-. Adım adım her görme işleminde bu işlem gerçekleşir böylece her görülen noktaya başlangıç noktasından olan mesafe hesaplanır. Fakat istisnai bir durum olan bir noktaya farklı yollardan daha kısa şekilde gitme kontrolüne de ihtiyaç duyulur. Bu işlem de eğer herhangi bir noktanın çevresinde görülmüş, kendisinden 2 derinlik ve ya daha fazla küçük bir nokta varsa o noktadan diğer noktalara dağılan tüm ağacın değerleri güncellenir.

  - Derinlik atanmış yollardan en kısa yolu bulma algoritması : Robot labirenti bitirip çıkışa ulaştığında. Robotun bildiği yollardan en kısası kullanıcıya kırmızı renkte gösterilir. En kısa yol bulunurken; Robotun bitiş noktasına atadığı derinlik üzerinden bir hesaplama yapılır. bu hesapta recursive bir fonksiyon kullanılır. Bitiş noktasından başlayarak bulunulan noktanın çevresindeki **Görülmüş yolların** robot tarafından atanmış derinliklerden en düşük derinlikli nokta için. Öncelikle Nokta bitişe giden yollar lisetesine eklenir, ardından çizimi yapılabilmesi indexlenir, eğer nokta başlangıç noktası değilse fonksiyon tekrar çağırılır ve tüm adımlar onun için de tekrar edilir. Eğer nokta başlangıç noktası ise bu yapı sonlanmalıdır en son olarak başalgıç noktası da bitişe giden yollara eklenir, indexlenir ve akış durdurulur.

- Url Okuma sistemi :
  Select URL sınıfının içerisindeki url entery veya Url1, Url2 buttonlarının herhangibirisinden gelen url tetikleme sinyali sonrasında Python **Requests** kütüphanesi kullanılarak öncelikle okuma işlemi yapılmaktadır. Ardından istenilen formatta grid kütüphanesinde işlenmektedir. sonrasında ise oyun ekranının taleb ettiğinden tüm çevresi duvarlarla kaplanıp Labirent Urlden oluşturulmaktadır.
- Labirent içerisindeki Algoritmalar :
  - Labirent oluşturma algoritması : Labirent oluştururken öncelikle istenilen boytlarda içerisi boş ve çevresi tümüyle duvarlarla dolu bir labirent oluşturulur. Ardından bu labirentin başlangıç noktasından, labirent oluştrmak için kurulmuş ve robotun haraketlerinde banzer davranan bir sistem bırakılır. Bu sistem çevresindeki noktaların durumuna göre farklı davranışlar izler. Eğer çevresindeki her yer önceden duvar atanmışsa çoktan kısırlaşmış demektir sistem biter. Eğer herhangi bir açık yol var ise bu açık yola gidip gitmemek için belirlenen şekide rastgelelik değeriyle 0 ya da 1 değeri üzerine zar atar. eğer değer 0 gelir ise labirente o nokta değiştirilemez ve yol olarak atanır. Eğer 1 gelirse de o nokta değiştirilemez ve duvar olarak atanır. Yol atandığı durumda aynı zamanda algoritma yol atanan nokta için tekrardan çağırılır. Bu sayede bitiş noktasına kadar ulaşan ve rastgele bitişe giden ve gitmeyen yollar içeren bir labirent oluşturulmuş olur.
    Rastgele oluşturulmuş 20x20 boyutunda labirent:
    
- Problem 1 En kısa yol algoritması: Problem 1 için istenen adım adım robottan bitişe giden en kısa yolun gösterilmesi için oluşturulmuş sistemdir. Robotun bitişe ulaştığında kullandığı algıritmaya benzerik göstermektedir. Robot her haraket ettiğinde güncellenir ve tekrardan çizilir. Robot ile tek bağlantısı robotun aktif bulunduğu noktadır, başka herhangi bir şeklide robota bağlı değildir. Oyun sınıfı tarafından hesaplanır ve çizilir. robottaki derinlik hesaplama algoritmasındaki görülüp görüşmeme koşulu kaldırılarak tüm labirentin recursive bir şekilde dolaşılıp derinlik atanmasına dayanır. Derinlikler atandıktan sonra bitiş noktasından başlanılarak yine robotun bitiş alogoritmasındali gibi çevresindeki en küçük noktaya gidilmesi üzerine kuruludur. fakat 0 değeri her derinlik güncellenmesinde başlangıç değil de aktif robot konumu olduğundan hesaplama sonucu robotun aktif konumu ile bitiş noktası arasındaki en kısa yol olacaktır.

IV. DENEYSEL SONUÇLAR

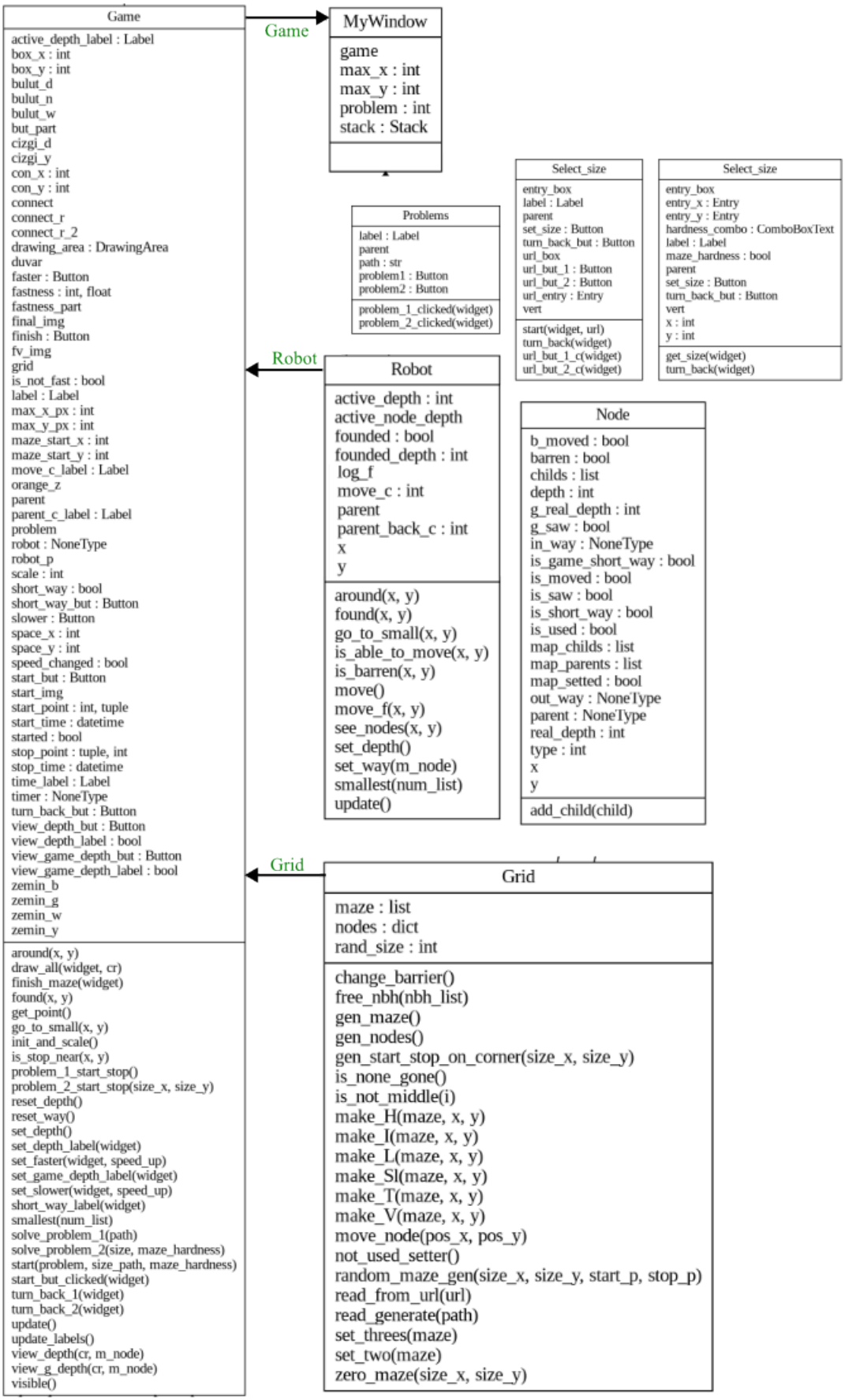
Nesneye yönelik programlama ve veri yapıları bilgilerinin kullanılarak hakkında herhangi bir bilgi sahibi olunmayan bir labirentin fahi çözülebileceği. Çözdükten sonra en kısa yolun tespit edilebileceği ve bu yolun birdahaki kullanımlar için öğrenilebileceği fark edilmiştir. Bir labirentin rasthele bir şekilde nasıl oluşturulabileceği ve bu yapı oluşturulurken hangi sistemlere ve programlama altyapısına ihtiyaç duyulduğu bilgilerine erişilmiştir.

V. SONUÇ

Nesneye yönelik programlama ve veri yapıları bilgilerinin kullanılarak bilmediği bir haritanda dolaşan ve sonucu bulduğunda haritanın bildiği kadarlık kısmını dikkate alarak en kısa yolu bulan bir robot sistemi oluşturulmuştur. Rastege bir şekilde bir labirent oluşturulmuş ve verilen url den txt dosyası okunarak labirente dönüştürülmüştür. ve robot tarafından bu labirentlerin çözüldüğü bir arayüz sistemi oluşturulmuştur. Bir oyunmuşcasında bu sistem kullanıcıya görselleştirilip sunulmuştur.

KAYNAKÇA

https://www.python.org/
https://python-gtk-3-tutorial.readthedocs.io/en/latest/
https://python-gtk-3-tutorial.readthedocs.io/en/latest/basics.html
https://python-gtk-3-tutorial.readthedocs.io/en/latest/application.html
https://python-gtk-3-tutorial.readthedocs.io/en/latest/entry.html
https://python-istihza.yazbel.com/


Game
active_depth_label : Label
box_x : int
box_y : int
bulut_d
bulut_n
bulut_w
but_part
cizgi_d
cizgi_y
con_x : int
con_y : int
connect
connect_r
connect_r_2
drawing_area : DrawingArea
duvar
faster : Button
fastness : int, float
fastness_part
final_img
finish : Button
fv_img
grid
is_not_fast : bool
label : Label
max_x_px : int
max_y_px : int
maze_start_x : int
maze_start_y : int
move_c_label : Label
orange_z
parent
parent_c_label : Label
problem
robot : NoneType
robot_p
scale : int
short_way : bool
short_way_but : Button
slower : Button
space_x : int
space_y : int
speed_changed : bool
start_but : Button
start_img
start_point : int, tuple
start_time : datetime
started : bool
stop_point : tuple, int
stop_time : datetime
time_label : Label
timer : NoneType
turn_back_but : Button
view_depth_but : Button
view_depth_label : bool
view_game_depth_but : Button
view_game_depth_label : bool
zemin_b
zemin_g
zemin_w
zemin_y
around(x, y)
draw_all(widget, cr)
finish_maze(widget)
found(x, y)
get_point()
go_to_small(x, y)
init_and_scale()
is_stop_near(x, y)
problem_1_start_stop()
problem_2_start_stop(size_x, size_y)
reset_depth()
reset_way()
set_depth()
set_depth_label(widget)
set_faster(widget, speed_up)
set_game_depth_label(widget)
set_slower(widget, speed_up)
short_way_label(widget)
smallest(num_list)
solve_problem_1(path)
solve_problem_2(size, maze_hardness)
start(problem, size_path, maze_hardness)
start_but_clicked(widget)
turn_back_1(widget)
turn_back_2(widget)
update()
update_labels()
view_depth(cr, m_node)
view_g_depth(cr, m_node)
visible()
Game
MyWindow
game
max_x : int
max_y : int
problem : int
stack : Stack
Problems
label : Label
parent
path : str
problem1 : Button
problem2 : Button
problem_1_clicked(widget)
problem_2_clicked(widget)
Select_size
entry_box
label : Label
parent
set_size : Button
turn_back_but : Button
url_box
url_but_1 : Button
url_but_2 : Button
url_entry : Entry
vert
start(widget, url)
turn_back(widget)
url_but_1_c(widget)
url_but_2_c(widget)
Select_size
entry_box
entry_x : Entry
entry_y : Entry
hardness_combo : ComboBoxText
label : Label
maze_hardness : bool
parent
set_size : Button
turn_back_but : Button
vert
x : int
y : int
get_size(widget)
turn_back(widget)
Robot
Robot
active_depth : int
active_node_depth
founded : bool
founded_depth : int
log_f
move_c : int
parent
parent_back_c : int
x
y
around(x, y)
found(x, y)
go_to_small(x, y)
is_able_to_move(x, y)
is_barren(x, y)
move()
move_f(x, y)
see_nodes(x, y)
set_depth()
set_way(m_node)
smallest(num_list)
update()
Node
b_moved : bool
barren : bool
childs : list
depth : int
g_real_depth : int
g_saw : bool
in_way : NoneType
is_game_short_way : bool
is_moved : bool
is_saw : bool
is_short_way : bool
is_used : bool
map_childs : list
map_parents : list
map_setted : bool
out_way : NoneType
parent : NoneType
real_depth : int
type : int
x
y
add_child(child)
Grid
Grid
maze : list
nodes : dict
rand_size : int
change_barrier()
free_nbh(nbh_list)
gen_maze()
gen_nodes()
gen_start_stop_on_corner(size_x, size_y)
is_none_gone()
is_not_middle(i)
make_H(maze, x, y)
make_I(maze, x, y)
make_L(maze, x, y)
make_Sl(maze, x, y)
make_T(maze, x, y)
make_V(maze, x, y)
move_node(pos_x, pos_y)
not_used_setter()
random_maze_gen(size_x, size_y, start_p, stop_p)
read_from_url(url)
read_generate(path)
set_threes(maze)
set_two(maze)
zero_maze(size_x, size_y)