МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНОГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЕ РЕСПУБЛИКИ УЗБЕКИСТАН

НАЦИОНАЛЬНЫЙ УНИВЕРСИТИТЕТ УЗБЕКИСТАНА ИМЕНИ МИРЗО УЛУГБЕКА

На правах рукописи

УДК 519.83

Жабборов Мухаммади Мусурмон уғли

“Топологичиские игры рассада и брюссельская капуста”

По специальности: 5А130101 – математика (Геометрия и топология)

ДИССЕРТАЦИЯ

На соискание академической степени магистра математических наук

Научный руководитель:

д.ф.-м.н.проф. М.Ш.Маматов

Ташкент 2017

Оглавление

ВВЕДЕНИЕ

Обоснование темы диссертации и его актуальность: Как отмечал наш президент Ш. Мирзияев в своей книге “Мы все вместе построим свободное, демократическое и процветающее государство Узбекистан”. Мы неуклонно и решительно продолжим государственную молодежную политику. И не только продолжим, но и поднимем ее на еще более высокий уровень в соответствии с требованиями сегодняшнего дня.

Мы мобилизуем все силы и возможности нашего государства и общества для того, чтобы наша молодежь обладала самостоятельным мышлением, высоким интеллектуальным и духовным потенциалом, ни в одной сфере не уступала своим сверстникам из других стран, была счастлива и уверена в своем будущем [1,40-ст].

Перед молодёжью стоит большая ответственность – получение знаний.

В результате развития фундаментальных направлений и в результате интеграции взаимо связанных направлений, появляются новые науки. Остановимся на теме диссертации, которая относится к отделу математики топологические игры.

Термин топологическая игра был впервые введен Берге [2], который определил основные идеи и формализм по аналогии с топологическими группами. Оказывается, что некоторые фундаментальные топологические конструкции имеют естественный аналог в топологических играх, примеры из них свойство Бэра, бэровские пространства, полнота и свойства сходимости, свойства разделения, покрытия, непрерывные образы, множества Суслина, и сингулярное разложение. В то же время, некоторые топологические свойства, которые возникают естественным образом в топологических играх могут быть обобщены за пределы теоретика - игровой контекст в силу этой двойственности, топологические

игры широко используются для описания новых свойств топологических пространств.

Краткое изложение методов исследования:

Проблема исследование решается теорией топологических игр и методами топологии.

Цель исследования:

-изучение топологических игр рассада и брюссельская капуста;

-обобщение топологических игр рассада и брюссельская капуста;

-исследование топологических игр рассада и брюссельская капуста на поверхности тора;

-создать программное обеспечение игр рассада и брюссельская капуста;

Задачи исследования:

- изучение историю игры “рассада” и её анализ;

- изучение игры “брюссельская капуста” и её анализ;

- исследовать обобщенную игру “рассада и брюссельская капуста”;

- изучение игры “брюссельская капуста” на поверхности тора;

- изучение игры “Обобщенная брюссельская капуста” на поверхности тора;

- создать программное обеспечение игры “рассада”;

- создать программное обеспечение игры “брюссельская капуста”;

Задачи исследования: Топологические игры “рассада” и брюссельская капуста, их характеристика на поверхности тора, также программное обеспечение этих игр.

Предмет исследования: Топологические игры “рассада” и “брюссельская капуста”.

Научная гипотеза исследования

Полученные результаты топологических игр «Обобщённая игра рассада и брюссельская капуста» являются общей для всех частных случаев.

Основные положения выносимые на защиту:

- обобщенные топологические игры “рассада” и “брюссельская капуста”;

- характеристика топологических игр “рассада” и “брюссельская капуста” на поверхности тора;

-программное обеспечение игры “рассада” и “брюссельская капуста”.

Научная и практическая значимость исследования: Теория топологических игр относится к наукам геометрия и топология, это исследования считается фундаментальным исследованием. Данная магистрская диссертация служит развитию теории топологических игр.

Публикации по исследованию:

Основные результаты исследования опубликованы на одном международном журнале и в трех научных сборниках республиканских конференций.

1-глава. “Топологичиские игры рассада и брюссельская капуста” на плоскости

1.1. История игры “Рассада” и её анализ

«Недавно мой друг, изучающий классическую филологию в Кембриджском университете, познакомил меня с игрой «РАССАДА». В течение последнего семестра на этой игре буквально помешался весь Кембридж. Некоторые особенности игры не лишены интереса с точки зрения топологии».

Так начиналось письмо, которое Мартин Гарднер получил в апреле 1967 году от одного студента-математика из Англии. Вскоре начали поступать и другие сообщения о том, что «Рассада» привилась и пышно расцвела на благодатной почве Кембриджа.

Заинтересованный новой игрой, Мартин Гарднер постарался разузнать как можно больше и о ней, и о ее создателях. Мои усилия увенчались успехом: мне удалось проследить историю игры до самых ее истоков. «Рассаду создали уже известный читателям профессор Джон Хортон Конуэй и кембриджский аспирант Майкл Стьюарт Патерсон.

Игра начинается с того, что на листе бумаги расставляют n точек (лунок для рассады). Анализ игры даже при $n=3$ оказывается намного более сложным, чем анализ игры в крестики и нолики, поэтому начинающим рекомендуется «выкапывать» в начале игры не больше 3-4 лунок. Делая очередной ход, играющий проводит линию («рассада пускает росток»), либо соединяющую одну точку с другой, либо описывающую

замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. Правила игры несложны. Их два:

1) линия может иметь любой вид, но не должна иметь точек самопересечения, пересекать ранее проведенные линии или проходить через ранее поставленные точки, не служащие ее началом и концом;

2) из каждой точки должно выходить не более трех линий.

Играющие по очереди проводят линии. В обычном варианте «рассады» победителем считается тот, кто сумеет провести последнюю линию. Возможен и другой вариант игры («мизер»), когда выигравшим считается тот, кто первым не сможет провести линию.

На рис. 1 показана типичная партия игры в «рассаду». Первый игрок выигрывает ее на седьмом ходу. (Ходы второго игрока показаны пунктирными линиями), чтобы их легче было отличать от ходов первого игрока. В обычной (непоказательной) игре не обязательно различать линии, проведенные первым и вторым игроками.

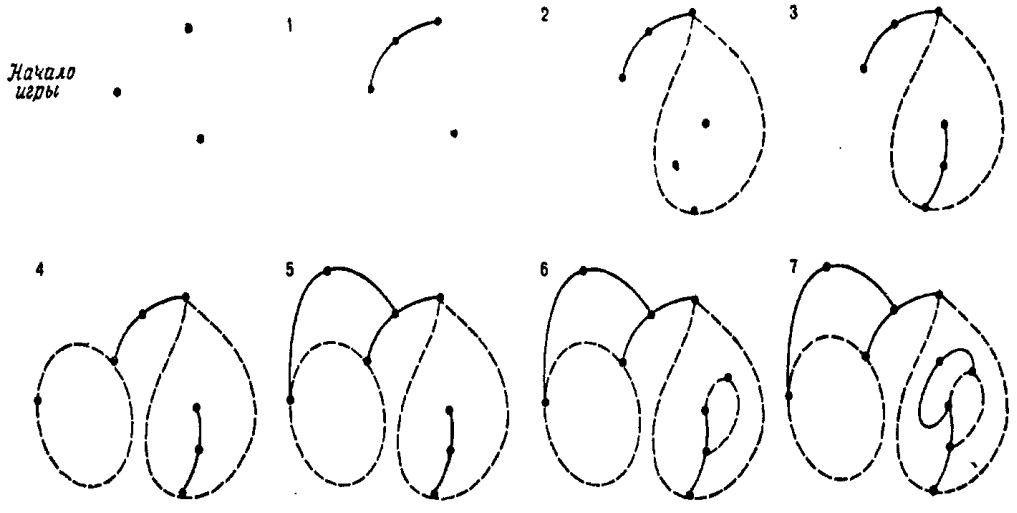


рисунок. 1

Игра получила свое название потому, что по мере развития партии линии разрастаются, подобно молодым побегам образуя фантастические узоры. Наиболее замечательная особенность игры состоит не в том, что «Рассада» принадлежит к числу комбинаторных игр (ибо таких игр много), а в том, что в ней используются топологические свойства плоскости.

Прибегнув к более строгой математической терминологии, можно сказать, что игра «Рассада» использует теорему Жордана о замкнутой кривой, которая гласит: «Всякая простая замкнутая кривая делит плоскость на две части: внутреннюю и внешнюю».

На первый взгляд может показаться, что «рассада» способна пускать ростки неограниченно долго, однако в действительности, как показал Конуэй, игра должна заканчиваться не более чем через $3n-1$ ходов. Из каждой лунки могут исходить три побега — три линии, которым по правилам игры разрешается пересекаться в одной точке. Лунка с тремя побегами называется отмершей, поскольку новых побегов из нее уже быть не может. Если в начальной позиции имелось n лунок, то число возможных побегов составляло $3n$. Каждый ход «убивает» два побега (один в начале и один в конце проводимой линии), но добавляет одну лунку, из которой может «проклюнуться» один росток. Следовательно, с каждым ходом возможное число побегов убывает на 1. Ясно, что продолжать игру при одном оставшемся побеге невозможно, поскольку каждый ход требует двух побегов. Таким образом, игра может продолжаться не более чем $3n-1$ ходов. В то же время, как нетрудно показать, игра не может закончиться раньше, чем через $2n$ ходов. Итак, если в начале игры было 3 лунки, то игра закончится не позже, чем на восьмом ходу, и не раньше, чем на шестом.

Доказательство того, что каждая партия в «Рассаду» не более чем $3n-1$ ходов.

Введем следующие обозначения: n-число начальных точек, m-число направлений, $L(m)$-число свободных направлений, то есть число выхода линий из любой точки, S-число неиспользованных точек по окончании игры.

Теорема. в игре “рассада” для **с**вободных направлений m начинающихся с n точек уместно следующее неравенство $m \le 3n-1$.

Для доказательства этой теоремы докажем следующие леммы.

Лемма 1. При выполнении m направлений число свободных направлений равно $L(m)=3n-m$.

Доказательство. выполним доказательство методом индукции.

При $m=0$ число свободных направлений будет $L(m)=3n$, это означает что в начале игры из n точек было 3 направленности.

Допустим $m=k$ и $L(k)=3n-k$.

Тогда при $m=k+1,$ докажем что $L(k+1)=3n-k-1.$ Изменение свободных направлений обозначим через ΔL, тогда $\Delta L=L(k+1)-L(k) \Leftrightarrow L(k+1)=L(k)+\Delta L$ уместно. Нам известно, что в игре каждое ребро связывает две открытые точки и порождает одну открытую точку, значит одно направление равно $\Delta L=-1$. Предположение верно для произвольного $m=k$, отсюда, $L(k+1)=L(k)+\Delta L=L(k)-1=3n-k-1=3n-(k+1)$ лемма доказана.

Лемма 2. Число свободных направлений $L(m)$ ограничено снизу 1, а также от $L(0)=3n$ до $L=S$ монотонно убывает. Здесь, S - число неиспользованных точек по окончании игры.

Доказательство. при $m=0$ число свободных направлений $L(0)=3n,$ и из-за $\Delta L=-1$выходит что оно уменьшается. $L(m)\neq 0$ потому что каждое ребро соединяет две точки и на этом ребре порождается еще одна точка. Отсюда, получаем что $L(m)\geq 1$.

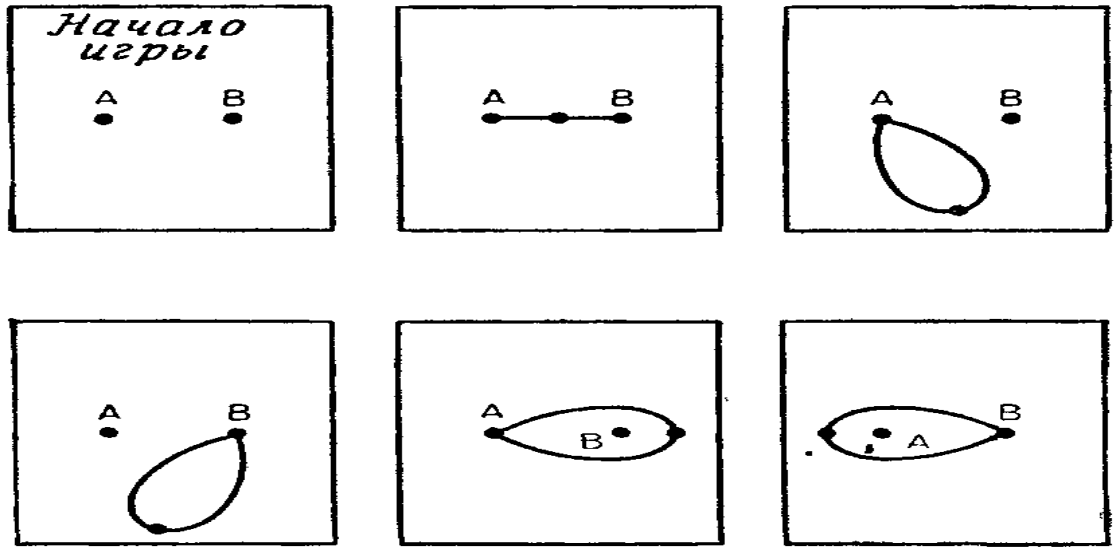
При помощи этих двух лемм доказываем теорему. $L(m)=3n-m$по лемме 1, а так же $L(m)\geq 1$по лемме 2. Значит,

$$L(m)=3n-m,\quad L(m)\geq 1 \quad\Rightarrow\quad 3n-m\geq 1 \quad\Rightarrow\quad m\leq 3n-1.$$

Теорема доказана.

Если в исходной позиции имеется лишь одна лунка, то игра тривиальна. У первого игрока нет выбора. Он должен провести замкнутую

петлю, начинающуюся и заканчивающуюся в единственной лунке. Если игра ведется, как обычно, то второй игрок выигрывает (в «мизерном» варианте он проигрывает), соединяя исходную и новую точки кривой, расположенной либо внутри, либо снаружи замкнутой петли. С точки зрения преимуществ, извлекаемых из ходов в игре, оба хода эквивалентны,



поскольку до того, как они сделаны, область, заключенная внутри замкнутой кривой, ничем не отличается от области, лежащей снаружи. Представим себе, что мы играем в «рассаду», проводя кривые на поверхности сферы. Проткнув поверхность сферы в точке, расположенной внутри замкнутой кривой, мы сможем растянуть ее на плоскость так, что все точки, лежавшие внутри замкнутой кривой, окажутся снаружи, а все точки, лежавшие снаружи, окажутся внутри. Топологическую эквивалентность областей, расположенных внутри и вне

рисунок. 2

Замкнутой кривой, не следует упускать из виду, поскольку она позволяет значительно упрощать анализ игры при числе лунок в начальной позиции, больше 2.

Если в начальной позиции имеются две лунки, игра сразу же приобретает интерес. Начнем с дебюта. Может показаться, будто у первого игрока имеется выбор из пяти ходов (рис. 2), однако не все эти ходы

различны. Из соображений симметрии ясно, что второе и третье (так же как и четвертое и пятое) начала партии эквивалентны. Кроме того, эквивалентность между внутренней и наружной областью замкнутой кривой позволяет считать равносильными, например, второе и четвертое, а следовательно, и все начала со второго по пятое. Таким образом, первый игрок должен выбрать лишь один из двух топологический различных ходов. Начертив дерево игры для всех возможных вариантов партий, нетрудно убедиться в том, что и при обычной, и при «мизерной» игре в «рассаду» с двумя лунками в исходной позиции второй игрок всегда может выиграть.

Анализируя игру с тремя лунками в исходной позиции, Конуэй показал, что в обычном варианте игры выигрыш всегда может остаться за первым, а в «мизерном» за вторым игроком. Д.Моллисон показал, что первый игрок всегда может выиграть в обычном вариант те игры, если в исходной позиции имеется 4 или 5 лунок. Поспорив с Конуэем на 10 шиллингов, что он успеет

Проанализировать игру с шестью лунками в начальной позиции в течение месяца, Моллисон представил трактат на 49 страницах, из его следовало, что выигрыш (в обычном варианте) должен остаться за вторым игроком. Доказательство оказалось столь громоздким, что до сих пор никому, кроме его автора, не удалось проследить ход рассуждений во всех деталях и убедиться в том, что в него нигде не вкралась ошибка. При игре в «мизерный» вариант с четыремя лунками в начальной позиции выигрывает второй игрок. Кто выигрывает в «мизерном» варианте в тех случаях, когда в начальной позиции имеется более четырех лунок, не известно. При семи и более лунках анализ игры становится столь сложным, что, по мнению Конуэя, требует, быстродействующей ЭВМ и хитроумной программы.

Таким образом, отыскание оптимальной стратегии для игры в «Рассаду» при произвольном числе лунок лежит за пределами

человеческих возможностей. Однако по мере приближения партии к концу нередко становится ясным, каким образом следует проводить замкнутые кривые, чтобы, разбивая плоскость на внешние и внутренние области, обеспечить себе выигрыш. Именно возможность такого разумного планирования делает «Рассаду» интеллектуальной забавой и позволяет игрокам совершенствовать мастерство от партии к партии. Вместе с тем «Рассада» изобилует неожиданностями, исключающими, по-видимому, самое существование универсальной стратегии, способной обеспечить выигрыш в любой ситуации. По оценке Конуэя, полный анализ игры с восемью лунками не под силу даже современным ЭВМ.

Известна точная дата рождения игры: она была изобретена во вторник 21 февраля 1967 г. В этот день Конуэй и Патерсон, покончив с чаепитием, сидели в гостиной математического факультета, безуспешно пытаясь придумать какую-нибудь новую игру, в которую можно было бы играть, вооружившись карандашом и бумагой. Конуэй разрабатывал вариант игры, в котором нужно было хитроумным способом складывать полоску марок, а Патерсон «переводил» все ходы на лаконичный язык графики. Было перебрано и отвергнуто множество различных правил, когда Патерсон вдруг заметил: «А почему бы нам не ставить на линии новую точку?»

«Как только новое правило было принято— вспоминает Конуэй, — все прочие правила тотчас же были отброшены как ненужные, начальная позиция упростилась и превратилась в набор из *n* точек. Словом, «Рассада» пустила корни».

Добавление новой точки имело столь решающее значение, что авторство, по единодушному мнению самих изобретателей «Рассады», было решено разделить в отношении 3:2 (3/5 получал Патерсон и 2/5- Конуэй).

«Были оговорены и другие, более сложные правила, позволяющие разделить любой, даже самый маленький лавровый листик, которым

потомки могут увенчать нас как изобретателей игры», – шутливо замечает Конуэй.

«На следующий день после того, как «Рассада» впервые пустила корни, продолжает он, в нее играли поголовно все. Во время завтрака всюду можно было видеть небольшие группки людей, смешно горбившихся над фантастическими узорами «Рассады». Некоторые пытались играть не только на приевшейся плоскости, но в поисках острых ощущений переходили на поверхность тора, бутылки Клейна и т. п. экзотику, а один математик даже подумывал об игре в «рассаду» в пространствах высших размерностей. Не остались в стороне от всеобщего увлечения и сотрудники секретариата.

Листы, исчерченные побегами, можно было обнаружить в самых неподходящих местах. Стоило мне начать объяснять кому-нибудь правила новой игры, как немедленно выяснялось, что мой собеседник уже наслышан о ней. Каналы, по которым распространялась информация, подчас были самыми «невероятными». Даже мои малолетние дочери, одной из которых исполнилось в ту пору три, а другой четыре года, играли в «Рассаду».

«Правда, добавляет Конуэй обычно мне удавалось у них выигрывать».

Название «Рассада» предложил Конуэй. Патерсон предлагал другое название «Корь», мотивируя его тем, что игра «прилипчива, как корь», и вспыхивает, как эпидемия, но привилась именно «Рассада».

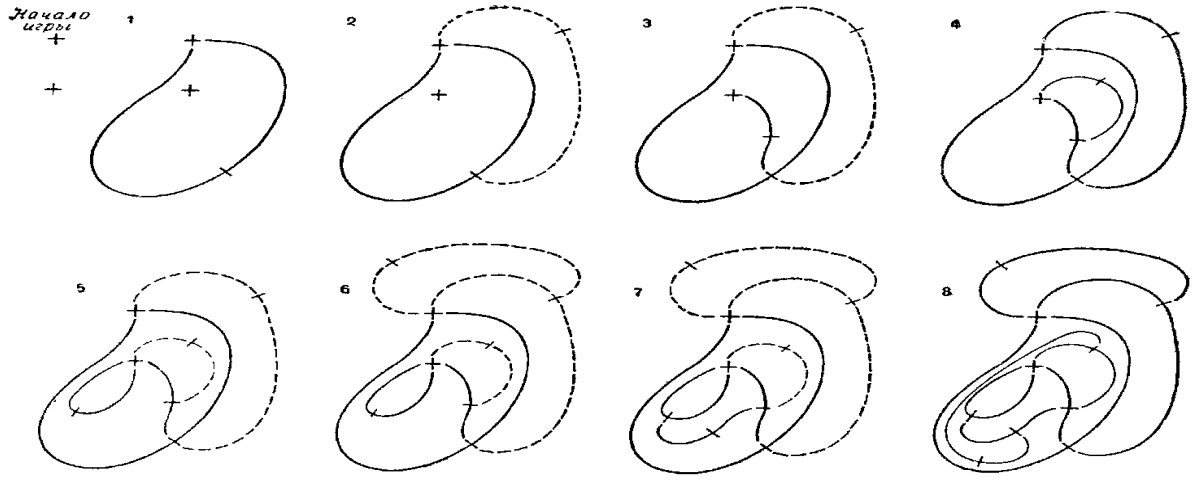
1.2. Игра “брюссельская капуста” и её анализ

Позднее Конуэй изобрел другую игру, внешне очень напоминающую «Рассаду». Желая подчеркнуть, что речь идет об игре-шутке, он назвал вторую игру «брюссельская капуста».

Вместо n точек начальная позиция при игре в «брюссельскую капусту» состоит из n крестиков. Очередной ход в «брюссельской капусте» состоит в том, что свободный конец одного из крестов соединяют со свободным концом другого и на проведенной кривой ставят поперечную черточку «перекладину» нового креста. Два других конца нового креста следует считать «отмершими», поскольку они уже «заняты» проведенной кривой, а использовать дважды одну и ту же оконечность креста нельзя. Так же как и в «Рассаде», кривые, проводимые при игре в «Брюссельскую капусту», не должны иметь точек самопересечения или пересекаться с ранее проведенными кривыми. Победителем в обычном варианте игры считается тот из игроков, кто делает последний ход (при игре в «мизерный» вариант, наоборот, тот, кто первым не сможет, сделать очередной ход).

После игры в «рассаду» «брюссельская капуста» может показаться более сложной и даже более изощренной игрой. Более того, поскольку каждый ход связывает кривой две оконечности крестов и порождает две новые оконечности, создается впечатление, будто игра может продолжаться бесконечно. Тем не менее все партии в «брюссельскую капусту» заканчиваются за конечное число ходов. Мы надеемся, что тот из читателей, кто сумеет полностью разобраться во всех тонкостях игры, по достоинству оценит юмор ее создателя. Чтобы облегчить первоначальное знакомство с новой игрой, приводим типичную партию на рис. 3. Второй игрок одерживает победу на восьмом ходу.

В письме, полученном мной от Конуэя, сообщалось о последних



успехах «огородничества». Патерсон и Конуэй научились оценивать конечную позицию «Рассады» по степени ее «вымирания» и разработали классификацию позиций нулевой степени. Таких позиций оказалось пять: блоха, жук, таракан, уховертка и скорпион. Мелкие блохи могут селиться иногда не по одиночке, а целыми гнездами на более крупных насекомых и скорпионах. Один из нарисованных Конуэем узоров представляет собой «не что иное, как вывернутую наизнанку уховертку внутри вывернутой наизнанку блохи». По сообщению

рисунок. 3

Конуэя, ему в соавторстве с Патерсоном удалось открыть весьма глубокую Фундаментальную Теорему о Вымирании Нулевой Степени ФТВНС. Так, семена, брошенные 21 февраля 1967 г., дали дружные всходы. Мы присутствуем при бурном развитии нового направления в занимательной математике — математического «огородничества»

Секрет игры-шутки «Брюссельская капуста» заключается в том, что в нее нельзя играть ни хорошо, ни плохо: каждая партия должна заканчиваться ровно через $5n-2$ ходов, где n - число крестиков в начальной позиции. В обычном варианте игры (когда выигравшим считается игрок, делающий последний ход) при нечетном числе крестиков в начальной позиции выигрывает первый игрок, при четном–второй. (В «мизерном» варианте ситуация противоположна.) Предложив кому-нибудь сыграть несколько партий в «Рассаду» игру, действительно требующую сообразительности и геометрической ситуации, можно затем «переключить» партнера на псевдо состязание в «Брюссельскую капусту». К его удивлению, вы всегда можете беспроигрышно заключать пари относительно того, кто выиграет очередную партию.

Доказательство того, что каждая партия в «брюссельскую капусту» заканчивается ровно за $5n-2$ ходов, я предоставляю ниже.

Теорема. Игра «брюссельская капуста» начинающаяся с n точек, заканчивается ровно через $5n-2$ ходов.

Доказательство. приведем следующие обозначения:

1. При начинании игры количество точек будет $V_{number}=n+m$, так как игра начинается с n точек – вершин и за m шагов добавляется m точек-вершин.

2. Ребер в графе получается $R_{number}=2m$ количество, так как каждая кривая порождает две новые оконечности.

3. В игре получается $G_{number}=4n$ количество граней графа.

Докажем 3-утверждение. В игре можно увидеть что в получаемом графе в конце игры получаются в каждой кривой есть точки в которых остаются неиспользованные линии выхода. Если обозначить их S, то $G_{number}=S$ будет, это значит что у каждой точки будет по одному неиспользованному выходу. Все возможные выходы линий из вершин обозначим $S(n)$, тогда $S(n)=4n+2m$ будет. Игра в 4 направленности из n точек при каждой следующей проведенной линии будет иметь 2 направленности. В игре каждая линия соединяет 2 открытые точки, это дает также уменьшение в 2 направленности. Из чего следует что в конце игры открытые направленности будут равны $S=S(n)-2m$. Значит, уместно следующее

$$S=S(n)-2m=4n+2m-2m=4n \;\Rightarrow\; S=G_{number}=4n$$

Теперь найдем связь между m и n. По теореме Эйлера имеем $V_{number}+G_{number}-R_{number}=2$. Значит,

$$V_{number}+G_{number}-R_{number}=2 \;\Rightarrow\; (n+m)+(4n)-(2m)=2 \;\Rightarrow\; 5n-m=2 \;\Rightarrow$$

$$\Rightarrow\; m=5n-2$$

Теорема доказана.

1.3. “Обобщеннаая игра рассада и брюссельская капуста”

Рассматривать обобщенные топологические игры «Рассада $n,4+3$», «Брюссельская капуста $n,3+4$», «Рассада $n_1,n_2,4+3$», «Брюссельская капуста $n_1,n_2,3+4$».

Обобщенная игра «Рассада $n,4+3$». «Рассада $n,4+3$» начинается с того, что на листе бумаги расставляют n точек, из каждой которых должно выходить не более четырех линий. Делая очередной ход, играющий проводит линию, либо соединяющую одну точку с другой, либо описывающую замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. Правила игры следующие: 1) линия может иметь любой вид, но не должна иметь точек самопересечения, пересекать ранее проведенные линии или проходить, через ранее поставленные точки, не служащие ее началом и концом; 2) из каждой новой проставленной точки должно выходить не более трех линий. Играющие по очереди проводят линии. Победителем считается тот, кто сумеет провести последнюю линию.

Для удобства введем следующие обозначения: n - точки в начальной позиции, m - число ходов, $L(n)$ - число свободных ходов, т.е. число возможных проводимых линий, S-число свободных направлений в конце игры. Нами доказана следующая теорема.

Теорема 1. Обобщенная игра «Рассада $n,4+3$» заканчивается не более чем через $4n-1$ ходов и не может закончиться раньше, чем $3n-1$ ходов, где n- число точек в начальной позиции.

Доказательство теоремы 1. Доказательство теоремы основывается на следующих лемм.

Лемма 1. Число свободных ходов игры через m ходов будет в виде $L(n)=4n-m$.

Доказательство леммы 1. Доказательство проводим с помощью метода математической индукции. При $m=0$, $L(n)=4n$ это означает, что в начале игры имеется $_n$ точек из каждого которых выходит четыре направления. Предположим, что при $m=k$ $L(n)=4n-k$ и покажем что при $m=k+1$ $L(n)=4n-(k+1)=4n-k-$. Для этого изменение числа свободных ходов за каждый ход обозначим через ΔL, тогда $\Delta L=L(k+1)-L(k)$. Ясно, что $\Delta L=-1$. Отсюда получим

$$L(k+1)=L(k)+\Delta L=4n-k-1.$$

Лемма доказана.

Лемма 2. Для число свободных ходов $L(m)$ игры имеет место неравенство $L(m)\geq 1$ и он от $L(0)=4n$ до $L=S$ монотонно убывает, где S- число свободных направлений оставшиеся в конце игры.

Доказательство леммы 2. При $m=0$, $L(0)=4n$ и из того, что $\Delta L=-1$ получим, что $L(m)$ монотонно убывает. $L(m)\neq 0$ так как каждый ход соединяет два направления и добавляет одно направление и значит $L(m)\geq 1$. Лемма доказана.

Теперь продолжим доказательство теоремы 1. По лемме 1 имеем $L(n)=4n-m$ и по лемме 2 получим $L(m)\geq 1$. И значит

$$L(n)=4n-m\geq 1,\ m\leq 4n-1$$

этим первая часть теоремы 1 доказана.

Лемма 3. Для числа грани графа, которое получено в конце игры имеет место неравенство $G_{number}\geq 2n+1$.

Доказательство леммы 3. Доказательство проводим с помощью метода математической индукции. При $n=1$ неравенство верно $G_{number}\geq 2$. Предположим, что неравенство при $n=k$ верно, т.е. $G_{number}\geq 2k+1$ и докажем, что оно верно при $n=k+1$ т.е. $G_{number}\geq 2(k+1)+1$. По предположению при $n=k$ для числа грани графа имеет место неравенство $G_{number}\geq 2k+1$, если из любой грани отмечая одну точку и продолжит игру,

то выходящей из этой точки три направления этой грани разделяет на три грани. Это означает, что число граней увеличивается на два, отсюда получим $G_{n+1} \geq G_{number} + 2 \geq 2k+1,\ G_{n+1} \geq 2k+1+2, G_{n+1} \geq 2(k+1)+1$.

Лемма доказана.

Лемма 4. Для числа грани графа, которое получено в игре через m ходов имеет место равенство $G_{number} = m - n + 2$.

Доказательство леммы 4. По правиле игры имеем $V_{number} = m + n$ и $R_{number} = 2m$. По теореме Эйлера $V_{number} + G_{number} - R_{number} = 2$, отсюда получим $m + n + G_{number} - 2m = 2$ и значит $G_{number} = m - n + 2$, что и требовалось доказать.

Вернемся еще раз доказательству теоремы 1. Используя леммы 3,4 получим $G_{number} \geq 2n+1$ и $G_{number} = m - n + 2$. Из них имеем $m - n + 2 \geq 2n+1,\ m \geq 3n-1.$ Значит, верно следующее неравенство $3n-1 \leq m \leq 4n-1.$ Теорема 1 доказана полностью.

Обобщенная игра «брюссельская капуста $n,3+4$***».*** Игра начинается с того, что на листе бумаги расставляют n точек, из каждой которых должно выходить не более трех линий. Делая очередной ход, играющий проводит линию, либо соединяющую одну точку с другой, либо описывающую замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. Правила игры следующие: 1) линия может иметь любой вид, но не должна иметь точек самопересечения, пересекать ранее проведенные линии или проходить, через ранее поставленные точки, не служащие ее началом и концом; 2) из каждой новой проставленной точки должно выходить не более четырех линий. Играющие по очереди проводят линии. Победителем считается тот, кто сумеет провести последнюю линию. Имеет место следующая теорема.

Теорема 2. Обобщенная игра «Брюссельская капуста $n,3+4$» заканчивается ровно через $4n-2$ ходов, где n- число точек в начальной позиции.

Доказательство теоремы 2. Докажем следующее утверждение:1) По условию игры в конце получается граф с вершинами $V_{number} = m+n$, так как игра начинается с n точек – вершин и за m шагов добавляется m точек-вершин; 2) В конце игры полученный граф имеет $R_{number} = 2m$ ребро, так как на каждом шаге добавляется два ребра; 3) В конце игры полученный граф имеет $G_{number} = 3m$ грани. Докажем последнее утверждение. В полученном графе каждой грани в некоторой точке остается единственное открытое направление. Если число открытых направлений обозначим через S, тогда число граней графа будет $G_{number} = S$. Если через $S(n)$ обозначим число всевозможных направлений, то $S(n) = 3n + 2m$, так как игра начинается с $3n$ направлением и за каждый шаг добавляется 2 направления. С другой стороны в каждом шаге соединяется 2 направления, значит $S = S(n) - 2m$. Отсюда имеем $S = S(n) - 2m = 3n + 2m - 2m = 3n,\ G_{number} = S = 3n.$ Теперь найдем связь между n и m. По теореме Эйлера имеет место следующее равенство $V_{number} + G_{number} - R_{number} = 2$.Отсюда получим $V_{number} + G_{number} - R_{number} = (n+m) + 3n - 2m = 4n - m = 2,\ m = 4n - 2.$

Теорема доказана полностью.

Обобщенная игра **«рассада** $n_1, n_2, 4+3$**».** На листе бумаги расставляем два типа точек: n_1 точек, из каждой которых должно выходить не более трех линий, n_2 точек, из каждой которых должно выходить не более четырех линий. Делая очередной ход, играющий проводит линию, либо соединяющую одну точку с другой, либо описывающую замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. Правила игры следующие: 1) линия может иметь любой вид, но не должна иметь точек самопересечения, пересекать ранее проведенные линии или проходить, через ранее поставленные точки, не служащие ее началом и концом; 2) из каждой новой проставленной точки

должно выходить не более трех линий. Играющие по очереди проводят линии. Победителем считается тот, кто сумеет провести последнюю линию. Нами доказана для этой игры следующая теорема.

Теорема 3. Обобщенная игра «рассада $n_1, n_2, 4+3$» заканчивается не более чем через $3n_1 + 4n_2 - 1$ ходов и не может закончиться раньше, чем $2(n_1 + n_2)$ ходов, где n_1, n_2 - число точек в начальной позиции.

Обобщенная игра **«брюссельская капуста** $n_1, n_2, 3+4$**».** На листе бумаги расставляем два типа точек: n_1 точек, из каждой которых должно выходить не более трех линий, n_2 точек, из каждой которых должно выходить не более четырех линий. Делая очередной ход в игре «Брюссельская капуста $n_1, n_2, 3+4$», играющий проводит линию, либо соединяющую одну точку с другой, либо описывающую замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. Из каждой новой проставленной точки должно выходить не более четырех линий. Так же как в «Рассаде», кривые, проводимые при игре в «Брюссельской капусте $n_1, n_2, 3+4$», не должны иметь точек самопересечения или пересекаться с ранее проведенными кривыми. Победителем игры считается тот из игроков, кто делает последний ход. Имеет место следующая теорема.

Теорема 4. Обобщенная игра «брюссельская капуста $n_1, n_2, 3+4$» заканчивается ровно через $4n_1 + 5n_2 - 2$ ходов, где n_1, n_2 - число точек в начальной позиции.

Доказательство теоремы 4. Докажем следующее утверждение:1) По условию игры в конце получается граф с вершинами $V_{number} = n_1 + n_2 + m$, так как игра начинается с $n_1 + n_2$ точек – вершин и за m шагов добавляется m точек-вершин; 2) В конце игры полученный граф имеет $R_{number} = 2m$ ребро, так как на каждом шаге добавляется два ребра; 3) В конце игры полученный граф имеет $G_{number} = 3n_1 + 4n_2$ грани. Докажем последнее утверждение. В

полученном графе каждой грани в некоторой точке остается единственное открытое направление. Если число открытых направлений обозначим через S, тогда число граней графа будет $G_{number} = S$. Если через $S(n)$ обозначим число всевозможных направлений, то $S(n) = 3n_1 + 4n_2 + 2m$, так как игра начинается с $3n_1 + 4n_2$ направлением и за каждый шаг добавляется 2 направления. С другой стороны в каждом шаге соединяется 2 направления, значит $S = S(n) - 2m$. Отсюда имеем $S = S(n) - 2m = 3n_1 + 4n_2 + 2m - 2m = 3n,\ G_{number} = S = 3n.$ Теперь найдем связь между $n_1 + n_2$ и m. По теореме Эйлера имеет место следующее равенство $V_{number} + G_{number} - R_{number} = 2$.Отсюда получим $V_{number} + G_{number} - R_{number} = (n_1 + n_2 + m) + 3n_1 + 4n_2 - 2m = 4n_1 + 5n_2 - m = 2,\ m = 4n_1 + 5n_2 - 2.$ Теорема доказана полностью.

2-глава. Топологичиские игры “брюссельская капуста” на поверхности тора

2.1. Игра “брюссельская капуста” на поверхности тора

Эта игра начинается с обозначения n точек на поверхности тора. В игре участвуют два игрока и каждый из них в очередном ходу или соединяет обозначенные две точки или из одного этих точек проводит замкнутую петлю, возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку.

Правила игры следующие:

1) линия может иметь любой вид, но не должна иметь точек самопересечения, пересекать ранее проведенные линии или проходить, через ранее поставленные точки, не служащие ее началом и концом;

2) из начальных точек и из каждой новой проставленной точки должно выходить не более четырех линий. Играющие по очереди проводят линии. Победителем считается тот, кто сумеет провести последнюю линию.

Нам известно, что путем склеивания прямого четырехугольника как в рисунке 1 можно создать тор.

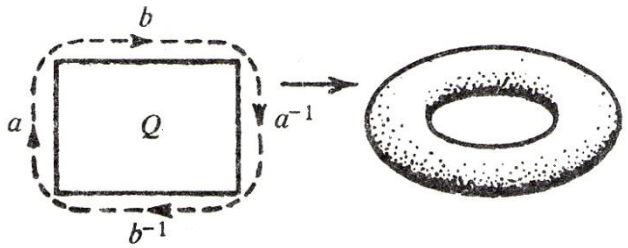


рисунок. 4

Начинать игру на поверхности тора сложнее, поэтому игру будем вести в прямом четырехугольнике, указанным выше.

Теорема 5. Для игры ***“брюссельская капуста”***, начинающаяся с n точек на поверхности тора всегда $5n-2 \le m \le 5n$, где

Доказательство теорема 5. первая часть теоремы ясно, то есть мы знаем, что $m = 5n - 2$. Потому что, если игра происходит в какой то закрытой части поверхности, это значит, игра происходит на плоскости (5-рисунок)

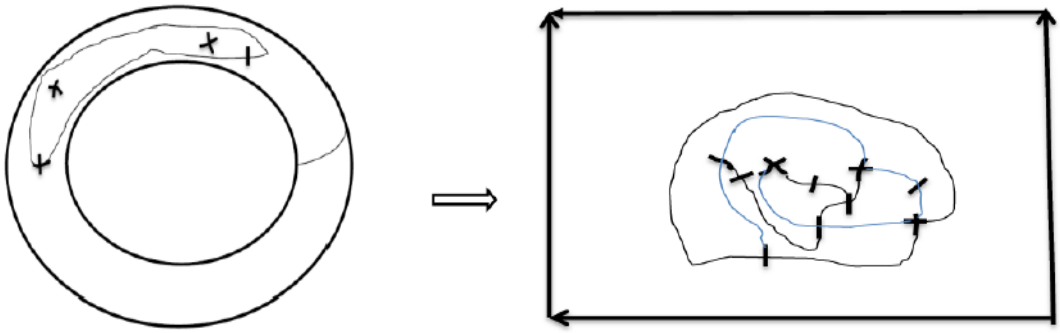


рисунок. 5

Нам известно, игра на плоскости заканчивается $m = 5n - 2$ шагов

Докажем вторую часть теоремы. Игру будем вести по всей поверхности тора (6-рисунок)

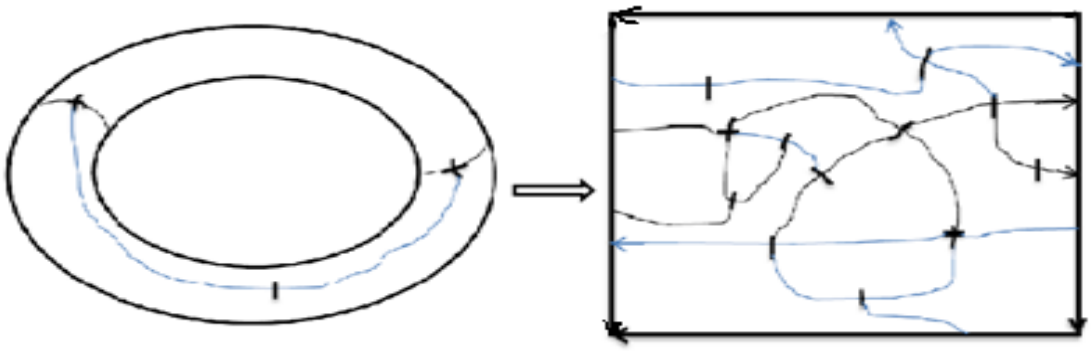


рисунок. 6

Ниже приведем известные утверждения:

1. По условию игры в конце получается граф с вершинами $V_{number} = m + n$, так как игра начинается с n точек – вершин и за m шагов добавляется m точек-вершин.

2. В конце игры полученный граф имеет $R_{number} = 2m$ ребро, так как на каждом шаге добавляется два ребра.

3. В конце игры полученный граф имеет $G_{number} = 4n$ грани.

Докажем 3 утверждение: В полученном графе каждой грани в некоторой точке остается единственное открытое направление. Если число

открытых направлений обозначим через S, тогда число граней графа будет $G_{number} = S$. Если через $S(n)$, обозначим число всевозможных направлений, то $S(n) = 4n + 2m$, так как игра начинается с $3n$ направлением и за каждый шаг добавляется 2 направления. С другой стороны в каждом шаге соединяется 2 направления, значит $S = S(n) - 2m$. Отсюда имеем $S = S(n) - 2m = 3n + 2m - 2m = 3n$, $G_{number} = S = 3n$. Теперь найдем связь между n и m. По теореме Эйлера для тора имеет место следующее равенство $V_{number} + G_{number} - R_{number} = 0$. Значит,

$$V_{number} + G_{number} - R_{number} = 0 \quad \Rightarrow \quad (n+m) + (4n) - (2m) = 0 \quad \Rightarrow$$

$$5n - m = 0 \quad \Rightarrow \quad \Rightarrow \quad m = 5n$$

Теорема доказана.

Итак, для числа m нижний рубеж $m = 5n - 2$, а верхний рубеж $m = 5n$, из этого следует $5n - 2 \le m \le 5n$.

Игра "брюссельская капуста" на поверхности тора в котором характеристика эйлера равна $\chi = 2 - 2g$. Теперь рассмотрим эту игру на поверхности тора, в котором характеристика Эйлера равна $\chi = 2 - 2g$

Теорема 6. Для игры, начинающееся с n точек на поверхности тора, в котором характеристика Эйлера равна $\chi = 2 - 2g$ всегда $5n - 2 \le m \le 5n + 2g - 2$

Доказательство теорема 6. Ясно, что $m = 5n - 2$ **,** потому что если игра происходит в какой то закрытой части поверхности, это значит игра происходит на плоскости. Нам известно, что на плоскости игра заканчивается через $m = 5n - 2$ ходов.

Находим верхний предел неравенства. Игру будем вести по всей поверхности тора, в которой характеристика Эйлера равна $\chi = 2 - 2g$.

Ниже приведем известные утверждения:

1. По условию игры в конце получается граф с вершинами $V_{number} = n + m$, так как игра начинается с n точек – вершин и за m шагов добавляется m точек-вершин.

2. В конце игры полученный граф имеет $R_{number} = 2m$ ребро, так как на каждом шаге добавляется два ребра.

3. В конце игры полученный граф имеет $G_{number} = 4n$ грани.

Докажем 3 утверждение: В полученном графе каждой грани в некоторой точке остается единственное открытое направление. Если число открытых направлений обозначим через S, тогда число граней графа будет $G_{number} = S$. Если через $S(n)$ обозначим число всевозможных направлений, то $S(n) = 4n + 2m$, так как игра начинается с $3n$ направлением и за каждый шаг добавляется 2 направления. С другой стороны в каждом шаге соединяется 2 направления, то есть убавляется 2 направления, значит $S = S(n) - 2m$. Отсюда имеем $S = S(n) - 2m = 4n + 2m - 2m = 4n \Rightarrow S = G_{number} = 4n$. Теперь найдем связь между n и m. По теореме Эйлера для тора имеет место следующее равенство $V_{number} + G_{number} - R_{number} = 2 - 2g$. Значит,

$$V_{number} + G_{number} - R_{number} = 2 - 2g \Rightarrow (n+m) + (4n) - (2m) = 2 - 2g \Rightarrow$$

$$5n - m = 2 - 2g \Rightarrow \Rightarrow m = 5n + 2g - 2$$

Итак, для числа m нижний предел $m = 5n - 2$, а верхний рубеж $m = 5n + 2g - 2$, из этого следует $5n - 2 \le m \le 5n + 2g - 2$

2.2. Обобщенная игра “брюссельская капуста” на поверхности тора

Эта игра начинается с обозначения на поверхности тора n_1 точек, из каждой которых должно выходить не более трех линий, n_2 точек, из каждой которых должно выходить не более четырех линий. В игре участвуют два игрока, делая очередной ход, играющий проводит линию, либо соединяющую одну точку с другой, либо описывающую замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. В ходе игры из каждой новой проставленной точки должно выходить не более четырех Играющие по очереди проводят линии. Победителем считается тот, кто сумеет провести последнюю линию.

Теорема 7. Для игры ***“Обобщенная брюссельская капуста“***, начинающаяся с $n = n_1 + n_2$ точек на поверхности тора всегда верно неравенство $4n_1 + 5n_2 - 2 \le m \le 4n_1 + 5n_2$.

Доказательство теорема 7. первая часть теоремы, то есть $m = 4n_1 + 5n_2 - 2$ ясно. Потому что, если игра происходит в какой то закрытой части поверхности, это значит игра происходит на плоскости и этот случай для плоскости мы доказывали.

Будем доказывать вторую часть теоремы. Игру будем вести по всей поверхности тора. Ниже приведем известные утверждения:

1. По условию игры в конце получается граф с вершинами $V_{number} = n_1 + n_2 + m$, так как игра начинается с $n_1 + n_2$ точек − вершин и за m шагов добавляется m точек-вершин.

2. В конце игры полученный граф имеет $R_{number} = 2m$ ребро, так как на каждом шаге добавляется два ребра.

3. В конце игры полученный граф имеет $G_{number} = 3n_1 + 4n_2$ грани.

Докажем 3 утверждение: В полученном графе каждой грани в некоторой точке остается единственное открытое направление. Если число открытых направлений обозначим через S, тогда число граней графа будет $G_{number} = S$. Если через $S(n_1, n_2)$, обозначим число всевозможных направлений, то $S(n_1, n_2) = 3n_1 + 4n_2 + 2m$, так как игра начинается с n_1 точек с тремя направлениями и с n_2 точек с четырьмя направлениями и за каждый шаг добавляется 2 направления. С другой стороны в каждом шаге соединяется 2 направления, это еще значит, что через каждое действие убавляется 2 направления. Итак, $S = S(n_1, n_2) - 2m$. Отсюда имеем

$$S = S(n_1, n_2) - 2m = 3n_1 + 4n_2 + 2m - 2m = 3n_1 + 4n_2 \Rightarrow S = G_{number} = 3n_1 + 4n_2$$

$$\Rightarrow \quad 4n_1 + 5n_2 - m = 0 \quad \Rightarrow \quad m = 4n_1 + 5n_2$$

Теперь найдем связь между n_1, n_2 и m. По теореме Эйлера имеет место следующее равенство $V_{number} + G_{number} - R_{number} = 0$. Значит,

$$V_{number} + G_{number} - R_{number} = 0 \quad \Rightarrow$$

$$(n_1 + n_2 + m) + (3n_1 + 4n_2) - (2m) = 0 \quad \Rightarrow$$

$$\Rightarrow \quad m = 5n$$

Итак, для числа m нижний рубеж $m = 4n_1 + 5n_2 - 2$, а верхний рубеж $m = 4n_1 + 5n_2$, из этого следует $4n_1 + 5n_2 - 2 \le m \le 4n_1 + 5n_2$

Могут быть следующие случаи:

1. $n_1 = 0$ в этом случае не могут быть точки из которых будут выходить по три линий, игра начинается с n_2 точек, из которых выходит не более четырех линий и это игра совпадает с игрой "**Брюссельская капуста на поверхности тора**", значит $5n_2 - 2 \le m \le 5n_2$

2. $n_2 = 0$ в этом случае не могут быть точки из которых будут выходить по четыре линий, игра начинается с n_1 точек, из которых выходит не более

трех линий и это игра совпадает с игрой “**Брюссельская капуста на поверхности тора**”, значит $4n_1 - 2 \le m \le 4n_1$.

Обобщенная игра “Брюссельская капуста” на поверхности тора в котором характеристика эйлера равна $\chi = 2 - 2g$**.** Эта игра начинается с обозначения на поверхности тора n_1 точек, из каждой которых должно выходить не более трех линий, n_2 точек, из каждой которых должно выходить не более четырех линий. В игре участвуют два игрока, делая очередной ход, играющий проводит линию, либо соединяющую одну точку с другой, либо описывающую замкнутую петлю и возвращающуюся в исходную точку, и затем ставит на проведенной линии новую точку. В ходе игры из каждой новой проставленной точки должно выходить не более четырех Играющие по очереди проводят линии. Победителем считается тот, кто сумеет провести последнюю линию.

Теорема 8. Для обобщенной игры «Брюссельская капуста», начинающееся с $n = n_1 + n_2$ точек на поверхности тора, в котором характеристика Эйлера равна $\chi = 2 - 2g$ всегда $4n_1 + 5n_2 - 2 \le m \le 4n_1 + 5n_2 + 2g - 2$

Доказательство теорема 8. Ясно, что нижний предел неравенства $m = 4n_1 + 5n_2 - 2$ потому что если игра происходит в какой то закрытой части поверхности, это значит игра происходит на плоскости. Этот случай для плоскости мы доказывали.

Находим верхний предел неравенства. Игру будем вести по всей поверхности тора, в которой характеристика Эйлера равна $\chi = 2 - 2g$.

Ниже приведем известные утверждения:

1. По условию игры в конце получается граф с вершинами $V_{number} = n_1 + n_2 + m$, так как игра начинается с $n_1 + n_2$ точек – вершин и за m шагов добавляется m точек-вершин.

2. В конце игры полученный граф имеет $R_{number} = 2m$ ребро, так как на каждом шаге добавляется два ребра.

3. В конце игры полученный граф имеет $G_{number} = 3n_1 + 4n_2$ грани.

Докажем 3 утверждение: В полученном графе каждой грани в некоторой точке остается единственное открытое направление. Если число открытых направлений обозначим через S, тогда число граней графа будет $G_{number} = S$. Если через $S(n_1, n_2)$ обозначим число всевозможных направлений, то $S(n_1, n_2) = 3n_1 + 4n_2 + 2m$, так как игра начинается с n_1 точек с тремя направлениями и с n_2 точек с четырмя направлениями и за каждый шаг добавляется 2 направления. С другой стороны в каждом шаге соединяется 2 направления, то есть убавляется 2 направления, значит $S = S(n_1, n_2) - 2m$. Отсюда имеем

$$S = S(n_1, n_2) - 2m = 3n_1 + 4n_2 + 2m - 2m = 3n_1 + 4n_2 \Rightarrow S = G_{number} = 3n_1 + 4n_2$$

Теперь найдем связь между n_1, n_2 и m. По теореме Эйлера для тора имеет место следующее равенство $V_{number} + G_{number} - R_{number} = 2 - 2g$. Значит,

$$V_{number} + G_{number} - R_{number} = 2 - 2g$$

$$(n_1 + n_2 + m) + (3n_1 + 4n_2) - (2m) = 2 - 2g \Rightarrow$$

$$\Rightarrow 4n_1 + 5n_2 - m = 2 - 2g \Rightarrow m = 4n_1 + 5n_2 + 2g - 2.$$

Итак, $4n_1 + 5n_2 - 2 \le m \le 4n_1 + 5n_2 + 2g - 2$

Могут быть следующие случаи:

1. $n_1 = 0$ в этом случае не могут быть точки из которых будут выходить по три линий, игра начинается с n_2 точек, из которых выходит не более четырех линий и это игра совпадает с игрой "**Брюссельская капуста на поверхности тора, характеристика которого равна** $\chi = 2 - 2g$ ", значит $5n_2 - 2 \le m \le 5n_2 + 2g - 2$

2. $n_2 = 0$ в этом случае не могут быть точки из которых будут выходить по четыре линий, игра начинается с n_1 точек, из которых выходит не более

трех линий и это игра совпадает с игрой “ **Брюссельская капуста** $n_1, n_2, 3+4$ **на поверхности тора, характеристика которого равна** $\chi = 2-2g$ ”, значит $4n_1 - 2 \le m \le 4n_1 + 2g - 2$

3-глава. Программное обеспечение игр “рассада и брюссельская капуста”

3.1. Программное обеспечение игры “рассада”

Мы на этом параграфе создадим программу для игры «Рассада». Для этого мы используем язык c#. Для создания программы для игры «Рассада» будем работать с пятью классами языка c#, они следующие:

cell.cs, circle.cs, form.cs, map.cs, path.cs

CELL.CS

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace WindowsFormsApplication1
{
   class Cell
   {
      public Point path0;
      public Point path1;
      public Point path2;

      public Circle circle;
      public Cell()
      {
         path0.X = 0;
         path1.X = 0;
         path2.X = 0;
         circle = null;
```

```
}
public void AddCircle(Circle c = null)
{
   this.circle = c;
}
public bool HasCircle()
{
   return this.circle != null;
}
public Circle GetCircle()
{
   return circle;
}
public void AddPath(int p, int pos)
{
   if (path0.X == 0) { path0.X = p; path0.Y = pos; }
   else
      if (path1.X == 0) { path1.X = p; path1.X = pos; }
      else
         if (path2.X == 0) { path2.X = p; path2.Y = pos; }
}
public bool HasWay()
{
   if (path0.X == 0) return true;
   if (HasCircle())
   {
      if (circle.hand == 3)
         return false;
   }
   else
      return false;
   if (path1.X == 0) return true;
   if (path2.X == 0) return true;
   return false;
 }
```

```
        public void setCircleHand()
        {
            circle.hand++;
        }
    }
}
```

CIRCLE.CS

```
using Microsoft.VisualBasic.PowerPacks;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WindowsFormsApplication1
{
    class Circle
    {
        private int radius;
        private Graphics objGraphics;
        private Point location;
        public int hand;


        public Circle(Form1 form1, int r, int nx, int ny)
        {
            hand = 0;
            location.X = nx;
            location.Y = ny;
            radius = r;
            objGraphics = form1.objGraphics;
            DrawCircle();
```

```
        }
        public void DrawCircle(){
            objGraphics.DrawEllipse(Pens.Red, location.X, location.Y, radius, radius);
        }
        public Point getLocation()
        {
            return this.location;
        }
    }
}
```

FORM.CS

```
using Microsoft.VisualBasic.PowerPacks;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Text;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public int rectangle_size;
        public Graphics objGraphics;
        public int circle_raduis;
        private int number_of_circle;
        private Circle[] circles;
```

```
private int horizontal_rectangle;
private int verical_rectangle;
private Random rnd = new Random();
private Path path;
private Map map;
private int step;
private bool comp_emeny;
private Path lastPossiblePath;
private bool hasPossiblePath;
bool blockUserMotion;
public Form1()
{
   InitializeComponent();
   /*
    * Installation of variable
    * windows size
    * rectanle size
    * canvas
    * path, path_length
    */
   step = 0;
   this.Height = 500;
   this.Width = 800;
   comp_emeny = false;
   rectangle_size = 20;
   horizontal_rectangle = this.Width / rectangle_size;
   verical_rectangle = this.Height / rectangle_size;
   circle_raduis = 20;
   number_of_circle = 0;
   circles = new Circle[100];
   map = new Map(this);
   objGraphics = this.CreateGraphics();
   objGraphics.SmoothingMode = SmoothingMode.AntiAlias;
   objGraphics.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;
   objGraphics.InterpolationMode = InterpolationMode.High;
```

```
            this.Height = 600;
            path = new Path(this);
        }
        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void Mouse_Move(object sender, MouseEventArgs e)
        {
            bool button_push = false;
            switch (e.Button)
            {
                case MouseButtons.Left:
                    if (blockUserMotion)
                        return;
                    button_push = true;
                    int i = e.Location.X / rectangle_size;
                    int j = e.Location.Y / rectangle_size;
                    if (path.checkPoint(i, j, this.map))
                    {
                        path.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2,
j * rectangle_size + (rectangle_size + 1) / 2));
                        path.setPointVal(i, j);
                    }
                    // Draw the points.
                    if (path.Points.Count < 2)
                        return;
                    // Draw the curve.
                    path.DrawPath();
                    break;

            }
            if (button_push)
            {
```

```
            textBox1.Text = "push";
        }
        else
        {
            textBox1.Text = "down";
            //this.Invalidate();
            if (path.Length() > 0)
            {
                if (path.CloseSession(rectangle_size, map, objGraphics))
                {
                    richTextBox1.AppendText("You have finished your attemp.\n");
                    richTextBox1.ScrollToCaret();
                    NextStep("Computer");

                                }
                path = new Path(this);
            }
        }
    }
    private void Form1_DoubleClick(object sender, EventArgs e)
    {
        //  Circle c = new Circle(this, (rectangle_size + 1) / 2, 100, 100);
    }
    private double destination(Point f, Point s)
    {
        return Math.Sqrt((f.X - s.X) * (f.X - s.X) + (f.Y - s.Y) * (f.Y - s.Y));
    }
    private Point generateLocation()
    {
        Point p = new Point();
        p.X = rnd.Next(3, horizontal_rectangle - 3) * rectangle_size;
        p.Y = rnd.Next(3, verical_rectangle - 3) * rectangle_size;
        return p;
    }
    private void button1_Click(object sender, EventArgs e)
```

```
{
    button1.Visible = false;
    comboBox1.Visible = false;
    comboBox3.Visible = false;
    number_of_circle = comboBox1.SelectedIndex + 1;
    if ((number_of_circle < 1) || (number_of_circle > 25))
        number_of_circle = 1;
    for (int i = 1; i <= number_of_circle; i++)
    {
        Point p = this.generateLocation();
        bool t = true;
        while (t)
        {
            t = false;
            for (int j = 1; j < i; j++)
            {
                if (destination(circles[j].getLocation(), p) < rectangle_size * 5) t = true;
            }
            if (t) p = this.generateLocation();
        }
        circles[i] = new Circle(this, circle_raduis, p.X, p.Y);
        map.AddCircle(circles[i], rectangle_size);
    }
    if (comboBox3.SelectedIndex == 1)
    {
        NextStep("Computer");
    }
    else
    {
        NextStep("User");
    }
            }
public int GetNumberOfCircle()
{
    return this.number_of_circle;
```

```
}
public void NextStep(String who)
{
   blockUserMotion = true;
   richTextBox1.AppendText("Looking for ways...\n");
   richTextBox1.ScrollToCaret();
   if (!CheckForHasWay())
   {
      richTextBox1.AppendText("There is no way to contiune game...\n");
      richTextBox1.AppendText("Game Over.\n");
      richTextBox1.ScrollToCaret();
      return;
   }
   richTextBox1.AppendText("There are several way to contiune game...\n");
   richTextBox1.ScrollToCaret();
   if (who == "User")
   {
      blockUserMotion = false;
   }
   else
   {
      blockUserMotion = true;
      Pen pen = Pens.Blue;
      lastPossiblePath.CloseSession(rectangle_size, map, objGraphics, pen);
      richTextBox1.AppendText("Computer finished attemp.\n");
      richTextBox1.ScrollToCaret();
      NextStep("User");
   }
   step++;
   if ((step % 2) == 0)
   {
      Gamer1.Text = "Gamer1";
      Gamer2.Text = "Gamer2<this>";
   }
   else
```

```
    {
        Gamer1.Text = "Gamer1<this>";
        Gamer2.Text = "Gamer2";
    }
}
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
}
public bool CheckForHasWay()
{
    List<Point> list = map.GetPossibleCircles();
    hasPossiblePath = false;
    if (list.Count == 0)
        return false;

    for (int i = 0; i < list.Count; i++)
    {
        hasPossiblePath = false;
        RecursionFindPath(list[i].X, list[i].Y);
        if (hasPossiblePath)
            return true;
    }
    return false;
}
private bool isOkPosition(int x, int y)
{
    if (x < 0) return false;
    if (y < 0) return false;
    if (x > 100) return false;
    if (y > 100) return false;
    return true;
}
private void RecursionFindPath(int x, int y)
{
    Cell[,] copy_map;
```

```
copy_map = map.getMap();
int[,] m = new int[200, 200];
for (int i = 0; i < 100; i++)
   for (int j = 0; j < 100; j++)
      m[i, j] = 0;
int count = 1;
m[x, y] = count;
bool loop = true;
bool found = false;
Point fountPoint = new Point();
Console.WriteLine("Try find way: point => " + x + ", " + y);
while (loop)
{
   loop = false;
   for (int i = 0; i < 100; i++)
   {
      for (int j = 0; j < 100; j++)
      {
         if (m[i, j] == count)
         {
            if (isOkPosition(i + 1, j))
            {
               if (map.HasWay(i + 1, j) && m[i + 1, j] == 0)
               {
                  loop = true;
                  m[i + 1, j] = count + 1;
                  if (map.isCircle(i + 1, j))
                  {
                     loop = false;
                     found = true;
                     fountPoint = new Point(i + 1, j);
                     break;
                  }
               }
            }
```

```
if (isOkPosition(i, j + 1))
{
   if (map.HasWay(i, j + 1) && m[i, j + 1] == 0)
   {
      loop = true;
      m[i, j + 1] = count + 1;
      if (map.isCircle(i, j + 1))
      {
         loop = false;
         found = true;
         fountPoint = new Point(i, j + 1);
         break;
      }
   }
}
if (isOkPosition(i - 1, j))
{
   if (map.HasWay(i - 1, j) && m[i - 1, j] == 0)
   {
      loop = true;
      m[i - 1, j] = count + 1;
      if (map.isCircle(i - 1, j))
      {
         loop = false;
         found = true;
         fountPoint = new Point(i - 1, j);
         break;
      }
   }
}
if (isOkPosition(i, j - 1))
{
   if (map.HasWay(i, j - 1) && m[i, j - 1] == 0)
   {
      loop = true;
```

```
                    m[i, j - 1] = count + 1;
                    if (map.isCircle(i, j - 1))
                    {
                        loop = false;
                        found = true;
                        fountPoint = new Point(i, j - 1);
                        break;
                    }
                }
            }
            if (isOkPosition(i - 1, j - 1))
            {
                if (map.HasWay(i - 1, j - 1) && m[i - 1, j - 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(-1, -1)))
                {
                    loop = true;
                    m[i - 1, j - 1] = count + 1;
                    if (map.isCircle(i - 1, j - 1))
                    {
                        loop = false;
                        found = true;
                        fountPoint = new Point(i - 1, j - 1);
                        break;
                    }
                }
            }
            if (isOkPosition(i + 1, j + 1))
            {
                if (map.HasWay(i + 1, j + 1) && m[i + 1, j + 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(1, 1)))
                {
                    loop = true;
                    m[i + 1, j + 1] = count + 1;
                    if (map.isCircle(i + 1, j + 1))
                    {
```

```
                    loop = false;
                    found = true;
                    fountPoint = new Point(i + 1, j + 1);
                    break;
                }
            }
        }
        if (isOkPosition(i - 1, j + 1))
        {
            if (map.HasWay(i - 1, j + 1) && m[i - 1, j + 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(-1, 1)))
            {
                loop = true;
                m[i - 1, j + 1] = count + 1;
                if (map.isCircle(i - 1, j + 1))
                {
                    loop = false;
                    found = true;
                    fountPoint = new Point(i - 1, j + 1);
                    break;
                }
            }
        }
        if (isOkPosition(i + 1, j - 1))
        {
            if (map.HasWay(i + 1, j - 1) && m[i + 1, j - 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(1, -1)))
            {
                loop = true;
                m[i + 1, j - 1] = count + 1;
                if (map.isCircle(i + 1, j - 1))
                {
                    loop = false;
                    found = true;
                    fountPoint = new Point(i + 1, j - 1);
```

```
                        break;
                    }
                }
            }

        }
        if (found)
            break;
    }
    count++;
}
Console.WriteLine();
for (int i = 0; i < 40; i++)
{
    for (int j = 0; j < 40; j++)
        Console.Write(m[i, j] + ", ");
    Console.WriteLine();
}
if (found)
{
    Path pp = new Path(this);
    int i = fountPoint.X;
    int j = fountPoint.Y;
    pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2, j *
rectangle_size + (rectangle_size + 1) / 2));
    pp.setPointVal(i, j);
    while (m[i, j] > 1)
    {
        if (i > 0 && m[i - 1, j] == (m[i, j] - 1))
        {
            i = i - 1;
            pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2, j
* rectangle_size + (rectangle_size + 1) / 2));
            pp.setPointVal(i, j);
```

```
                }
                else
                    if (j > 0 && m[i, j - 1] == (m[i, j] - 1))
                    {
                        j = j - 1;
                        pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2,
j * rectangle_size + (rectangle_size + 1) / 2));
                        pp.setPointVal(i, j);
                    }
                    else
                        if (m[i + 1, j + 1] == (m[i, j] - 1) )
                        {
                            i++;
                            j++;
                            pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) /
2, j * rectangle_size + (rectangle_size + 1) / 2));
                            pp.setPointVal(i, j);
                        }
                        else
                            if (i > 0 && j > 0 && m[i - 1, j - 1] == (m[i, j] - 1) )
                            {
                                i--;
                                j--;
                                pp.Points.Add(new Point(i * rectangle_size + (rectangle_size +
1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                                pp.setPointVal(i, j);
                            }
                            else
                                if (m[i + 1, j] == (m[i, j] - 1))
                                {
                                    i++;
                                    pp.Points.Add(new Point(i * rectangle_size + (rectangle_size
+ 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                                    pp.setPointVal(i, j);
                                }
```

```
                    else
                        if (m[i, j + 1] == (m[i, j] - 1))
                        {
                            j++;
                            pp.Points.Add(new Point(i * rectangle_size +
(rectangle_size + 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                            pp.setPointVal(i, j);
                        }
                        else
                            if (j > 0 && m[i + 1, j - 1] == (m[i, j] - 1))
                            {
                                i++;
                                j--;
                                pp.Points.Add(new Point(i * rectangle_size +
(rectangle_size + 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                                pp.setPointVal(i, j);
                            }
                            else
                                if (i > 0 && m[i - 1, j + 1] == (m[i, j] - 1))
                                {
                                    i--;
                                    j++;
                                    pp.Points.Add(new Point(i * rectangle_size +
(rectangle_size + 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                                    pp.setPointVal(i, j);
                                }

                }
                hasPossiblePath = true;
                lastPossiblePath = pp;
                return;
            }

        }
```

```
        private void button2_Click(object sender, EventArgs e)
        {
            lastPossiblePath.DrawPath();
        }
    }
}
```

MAP.CS

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    class Map
    {
        public Cell[,] map;
        private int number_of_line;
        private List<Path> pathes;
        private int number_of_circle;
        public List<Circle> circles;
        Form1 form1;
        public Map(Form1 form)
        {
            form1 = form;
            map = new Cell[1000, 1000];
            for (int i = 0; i < 1000; i++)
                for (int j = 0; j < 1000; j++)
                    map[i, j] = new Cell();
```

```
            number_of_line = 999;
            number_of_circle = 0;
            pathes = new List<Path>();
            circles = new List<Circle>();
        }
        public void AddLine(Path path)
        {
            // lines add with index from 1000 to infinity
            number_of_line++;
            pathes.Add(path);
            for (int i = 0; i < (path.ArrayPoints.Count); i++)
                map[path.ArrayPoints[i].X,
path.ArrayPoints[i].Y].AddPath(number_of_line, i);
            int ii = path.ArrayPoints[(path.ArrayPoints.Count - 1) / 2].X;
            int jj = path.ArrayPoints[(path.ArrayPoints.Count - 1) / 2].Y;
            Circle c = new Circle(form1, form1.circle_raduis, ii * form1.rectangle_size, jj
* form1.rectangle_size);
            c.DrawCircle();
            c.hand = 2;
            AddCircle(c, form1.rectangle_size);
        }
        public void AddCircle(Circle c, int rectangle_size)
        {
            // circles add index from 1 to 999
            number_of_circle++;
            int i = c.getLocation().X / rectangle_size;
            int j = c.getLocation().Y / rectangle_size;
            map[i, j].AddCircle(c);
            circles.Add(c);
        }
        public bool isCircle(int i, int j)
        {
            return map[i, j].HasCircle();
        }
        public Circle getCircleFromMap(int i, int j)
```

```
        {
            return map[i, j].GetCircle();
        }
        public void Redraw()
        {
            for (int i = 0; i < circles.Count; i++)
                circles[i].DrawCircle();
            for (int i = 0; i < pathes.Count; i++)
            {
                pathes[i].DrawPath();
            }
        }
        public bool HasWay(int x, int y)
        {
            return map[x, y].HasWay();
        }
        public bool CheckCellsToConnection(Point p, Point move)
        {
            //map[p.X + move.X, p.Y];
            //map[p.X, p.Y + move.Y];
            //Console.WriteLine(p.X + ", " + p.Y);
            if ((p.X + move.X) < 0) return true;
            if ((p.Y + move.Y) < 0) return true;

            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path0, map[p.X,
p.Y + move.Y].path0))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path0, map[p.X,
p.Y + move.Y].path1))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path0, map[p.X,
p.Y + move.Y].path2))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path1, map[p.X,
p.Y + move.Y].path0))
```

```
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path1, map[p.X,
p.Y + move.Y].path1))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path1, map[p.X,
p.Y + move.Y].path2))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path2, map[p.X,
p.Y + move.Y].path0))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path2, map[p.X,
p.Y + move.Y].path1))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path2, map[p.X,
p.Y + move.Y].path2))
                return true;


            return false;
        }
        public bool CheckTwoCellsForConnection(Point p1, Point p2)
        {
            if (p1.X == p2.X && (Math.Abs(p1.Y - p2.Y) == 1))
                return true;
            return false;
        }
        public List<Point> GetPossibleCircles()
        {
            List<Point> list = new List<Point>();
            for (int i = 0; i < 1000; i++)
                for (int j = 0; j < 1000; j++)
                {
                    if (map[i, j].HasCircle() && map[i,j].HasWay())
                    {
                        list.Add(new Point(i,j));
```

```
                }
            }
            return list;
        }
        public Cell[,] getMap()
        {
            return map;
        }
    }
}
```

PATH.CS

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    class Path
    {
        public int[,] path;
        private int path_length;
        public List<Point> Points;
        public List<Point> ArrayPoints;
        public Point FirstPoint;
        public Point LastPoint;
        private Graphics objGraphics;
        private Pen pen = Pens.Red;

        public void Clone(Path p)
```

```
        {
            this.path = p.path;
            this.path_length = p.path_length;
            this.Points = p.Points;
            this.ArrayPoints = p.ArrayPoints;
            this.FirstPoint = p.FirstPoint;
            this.LastPoint = p.LastPoint;
            this.objGraphics = p.objGraphics;

        }
        public Path(Form1 form)
        {
            Points = new List<Point>();
            ArrayPoints = new List<Point>();
            path = new int[100, 100];
            path_length = 0;
            objGraphics = form.objGraphics;
        }
        public void Reset()
        {
            Points = new List<Point>();
            ArrayPoints = new List<Point>();
            path = new int[1000, 1000];
            path_length = 0;
        }
        public bool SelfCheck(int x, int y)
        {

            if (x < 0 || y < 0)
                return false;
            if (path_length == 0) return true;
            Point p = new Point(x - LastPoint.X, y - LastPoint.Y);
            //Console.WriteLine(p.X+" , "+p.Y);
            if (Math.Abs(p.X + p.Y) == 1 && p.X > -2 && p.X < 2 && p.Y < 2 && p.Y > -
2) return true;
```

```
            //dioganal harakat
            if (Math.Abs(p.X) == 1 && Math.Abs(p.Y) == 1)
                return Math.Abs(path[LastPoint.X + p.X, LastPoint.Y] - path[LastPoint.X,
LastPoint.Y + p.Y]) != 1;

            return false;
        }
        public bool CheckWithMap(int x, int y, Map map)
        {
            if (path_length < 2)
                if (map.isCircle(x, y)) return true;
            Point p = new Point(x - LastPoint.X, y - LastPoint.Y);
            //dioganal harakat
            if (x > 0 && y > 0)
                if (map.isCircle(LastPoint.X, LastPoint.Y) && path_length > 1) return
false;

            if (Math.Abs(p.X) == 1 && Math.Abs(p.Y) == 1)
                return !map.CheckCellsToConnection(LastPoint, p) && map.HasWay(x,
y);

            return map.HasWay(x, y);
        }
        public bool checkPointForZero(int i, int j)
        {
            if (i < 0 || j < 0) return false;
            return this.path[i, j] == 0;

        }
        public bool checkPoint(int x, int y, Map map)
        {
            if (!map.isCircle(LastPoint.X, LastPoint.Y) && FirstPoint.X == x &&
FirstPoint.Y == y && path_length > 2)
            {
                if (map.map[x, y].circle.hand < 2)
```

```
            return true;
        else
            return false;
    }

    return checkPointForZero(x, y) && SelfCheck(x, y) && CheckWithMap(x, y,
map);
}

public void setPointVal(int i, int j)
{
    ArrayPoints.Add(new Point(i, j));
    if (path_length == 0)
        FirstPoint = new Point(i, j);
    path_length++;
    path[i, j] = path_length;
    LastPoint = new Point(i, j);
}
public bool CanAddLine(Map map)
{

    bool stat = true;
    if (!map.isCircle(FirstPoint.X, FirstPoint.Y))
    {
        stat = false;
    }
    else
    {
        if (map.getCircleFromMap(FirstPoint.X, FirstPoint.Y).hand > 2)
            stat = false;
    }

    if (!map.isCircle(LastPoint.X, LastPoint.Y))
    {
        stat = false;
```

```
            }
            else
            {
                if (map.getCircleFromMap(LastPoint.X, LastPoint.Y).hand > 2)
                    stat = false;
            }


            if (LastPoint == FirstPoint)
            {
                if (stat)
                    if (map.getCircleFromMap(LastPoint.X, LastPoint.Y).hand > 1)
                        stat = false;
            }
            return stat;
        }
        public bool CloseSession(int rectangle_size, Map map, Graphics objGraphics,
Pen p = null)
        {
            if (p != null)
                pen = p;
            else
                pen = Pens.Red;

            bool ans = false;
            if (Points.Count < 2)
                return ans;
            if (CanAddLine(map))
            {
                map.map[FirstPoint.X, FirstPoint.Y].setCircleHand();
                map.map[LastPoint.X, LastPoint.Y].setCircleHand();
                ans = true;
                map.AddLine(this);
            }
            objGraphics.Clear(Color.White);
```

```
            map.Redraw();
            return ans;
        }
        public int Length()
        {
            return path_length;
        }
        public void DrawPath()
        {
            if (Points.Count < 2)
                return;
            objGraphics.DrawCurve(pen, Points.ToArray());
        }
        public Path getThis()
        {
            return this;
        }
    }
}
```

3.2. Программное обеспечение игры “ Брюссельская капуста”

Мы на этом параграфе создадим программу для игры “Брюссельская капуста”. Для этого мы используем язык c#. Для создания программы для игры “Брюссельская капуста” будем работать с пятью классами языка c#, они следующие:

cell.cs, circle.cs, form.cs, map.cs, path.cs

CELL.CS

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace WindowsFormsApplication1
{
   class Cell
   {
      public Point path0;
      public Point path1;
      public Point path2;
      public Point path3;
      public Circle circle;
      public Cell()
      {
         path0.X = 0;
         path1.X = 0;
         path2.X = 0;
         path3.X = 0;
         circle = null;
      }
      public void AddCircle(Circle c = null)
      {
```

```
    this.circle = c;
}
public bool HasCircle()
{
    return this.circle != null;
}
public Circle GetCircle()
{
    return circle;
}
public void AddPath(int p, int pos)
{
    if (path0.X == 0) { path0.X = p; path0.Y = pos; }
    else
        if (path1.X == 0) { path1.X = p; path1.X = pos; }
        else
            if (path2.X == 0) { path2.X = p; path2.Y = pos; }
            else
                if (path3.X == 0) { path3.X = p; path3.Y = pos; }
}
public bool HasWay()
{
    if (path0.X == 0) return true;
    if (HasCircle())
    {
        if (circle.hand == 4)
            return false;
    }
    else
        return false;
    if (path1.X == 0) return true;
    if (path2.X == 0) return true;
    if (path3.X == 0) return true;
    return false;
 }
```

```
        public void setCircleHand()
        {
            circle.hand++;
        }
    }
}
```

CIRCLE.CS

```
using Microsoft.VisualBasic.PowerPacks;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace WindowsFormsApplication1
{
    class Circle
    {
        private int radius;
        private Graphics objGraphics;
        private Point location;
        public int hand;

        public Circle(Form1 form1, int r, int nx, int ny)
        {
            hand = 0;
            location.X = nx;
            location.Y = ny;
            radius = r;
            objGraphics = form1.objGraphics;
            DrawCircle();

        }
       public void DrawCircle(){
            objGraphics.DrawEllipse(Pens.Red, location.X, location.Y, radius, radius);
```

```
        }
        public Point getLocation()
        {
            return this.location;
        }
    }
}
```

FORM1.CS

```
using Microsoft.VisualBasic.PowerPacks;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Text;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public int rectangle_size;
        public Graphics objGraphics;
        public int circle_raduis;
        private int number_of_circle;
        private Circle[] circles;
        private int horizontal_rectangle;
        private int verical_rectangle;
        private Random rnd = new Random();
        private Path path;
        private Map map;
        private int step;
```

```
private bool comp_emeny;
private Path lastPossiblePath;
private bool hasPossiblePath;
bool blockUserMotion;
public Form1()
{
   InitializeComponent();
   /*
    * Installation of variable
    * windows size
    * rectanle size
    * canvas
    * path, path_length
    */
   step = 0;
   this.Height = 500;
   this.Width = 800;
   comp_emeny = false;
   rectangle_size = 20;
   horizontal_rectangle = this.Width / rectangle_size;
   verical_rectangle = this.Height / rectangle_size;
   circle_raduis = 20;
   number_of_circle = 0;
   circles = new Circle[100];
   map = new Map(this);
   objGraphics = this.CreateGraphics();
   objGraphics.SmoothingMode = SmoothingMode.AntiAlias;
   objGraphics.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;
   objGraphics.InterpolationMode = InterpolationMode.High;
   this.Height = 600;
   path = new Path(this);
}
private void Form1_Load(object sender, EventArgs e)
{
```

```
        }

        private void Mouse_Move(object sender, MouseEventArgs e)
        {
            bool button_push = false;
            switch (e.Button)
            {
                case MouseButtons.Left:
                    if (blockUserMotion)
                        return;
                    button_push = true;
                    int i = e.Location.X / rectangle_size;
                    int j = e.Location.Y / rectangle_size;
                    if (path.checkPoint(i, j, this.map))
                    {
                        path.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2,
j * rectangle_size + (rectangle_size + 1) / 2));
                        path.setPointVal(i, j);
                    }
                    // Draw the points.
                    if (path.Points.Count < 2)
                        return;
                    // Draw the curve.
                    path.DrawPath();
                    break;
            }
            if (button_push)
            {
                textBox1.Text = "push";
            }
            else
            {
                textBox1.Text = "down";
                //this.Invalidate();
                if (path.Length() > 0)
```

```
        {
            if (path.CloseSession(rectangle_size, map, objGraphics))
            {
                richTextBox1.AppendText("You have finished your attemp.\n");
                richTextBox1.ScrollToCaret();
                NextStep("Computer");

                            }
            path = new Path(this);
        }
    }
}
private void Form1_DoubleClick(object sender, EventArgs e)
{
    //  Circle c = new Circle(this, (rectangle_size + 1) / 2, 100, 100);
}
private double destination(Point f, Point s)
{
    return Math.Sqrt((f.X - s.X) * (f.X - s.X) + (f.Y - s.Y) * (f.Y - s.Y));
}
private Point generateLocation()
{
    Point p = new Point();
    p.X = rnd.Next(3, horizontal_rectangle - 3) * rectangle_size;
    p.Y = rnd.Next(3, verical_rectangle - 3) * rectangle_size;
    return p;
}
private void button1_Click(object sender, EventArgs e)
{
    button1.Visible = false;
    comboBox1.Visible = false;
    comboBox3.Visible = false;
    number_of_circle = comboBox1.SelectedIndex + 1;
    if ((number_of_circle < 1) || (number_of_circle > 25))
        number_of_circle = 1;
```

```
    for (int i = 1; i <= number_of_circle; i++)
    {
        Point p = this.generateLocation();
        bool t = true;
        while (t)
        {
            t = false;
            for (int j = 1; j < i; j++)
            {
                if (destination(circles[j].getLocation(), p) < rectangle_size * 5) t = true;
            }
            if (t) p = this.generateLocation();
        }
        circles[i] = new Circle(this, circle_raduis, p.X, p.Y);
        map.AddCircle(circles[i], rectangle_size);
    }
    if (comboBox3.SelectedIndex == 1)
    {
        NextStep("Computer");
    }
    else
    {
        NextStep("User");
    }

}
public int GetNumberOfCircle()
{
    return this.number_of_circle;
}
public void NextStep(String who)
{
    blockUserMotion = true;
    richTextBox1.AppendText("Looking for ways...\n");
    richTextBox1.ScrollToCaret();
```

```
    if (!CheckForHasWay())
    {
        richTextBox1.AppendText("There is no way to contiune game...\n");
        richTextBox1.AppendText("Game Over.\n");
        richTextBox1.ScrollToCaret();
        return;
    }
    richTextBox1.AppendText("There are several way to contiune game...\n");
    richTextBox1.ScrollToCaret();
    if (who == "User")
    {
        blockUserMotion = false;
    }
    else
    {
        blockUserMotion = true;
        Pen pen = Pens.Blue;
        lastPossiblePath.CloseSession(rectangle_size, map, objGraphics, pen);
        richTextBox1.AppendText("Computer finished attemp.\n");
        richTextBox1.ScrollToCaret();
        NextStep("User");
    }
    step++;
    if ((step % 2) == 0)
    {
        Gamer1.Text = "Gamer1";
        Gamer2.Text = "Gamer2<this>";

    }
    else
    {
        Gamer1.Text = "Gamer1<this>";
        Gamer2.Text = "Gamer2";
    }
}
```

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
}
public bool CheckForHasWay()
{
   List<Point> list = map.GetPossibleCircles();
   hasPossiblePath = false;
   if (list.Count == 0)
      return false;
   for (int i = 0; i < list.Count; i++)
   {
      hasPossiblePath = false;
      RecursionFindPath(list[i].X, list[i].Y);
      if (hasPossiblePath)
         return true;
   }
   return false;
}
private bool isOkPosition(int x, int y)
{
   if (x < 0) return false;
   if (y < 0) return false;
   if (x > 100) return false;
   if (y > 100) return false;
   return true;
}
private void RecursionFindPath(int x, int y)
{
   Cell[,] copy_map;
   copy_map = map.getMap();
   int[,] m = new int[200, 200];
   for (int i = 0; i < 100; i++)
      for (int j = 0; j < 100; j++)
         m[i, j] = 0;
```

```
int count = 1;
m[x, y] = count;
bool loop = true;
bool found = false;
Point fountPoint = new Point();
Console.WriteLine("Try find way: point => " + x + ", " + y);
while (loop)
{
   loop = false;
   for (int i = 0; i < 100; i++)
   {
      for (int j = 0; j < 100; j++)
      {
         if (m[i, j] == count)
         {
            if (isOkPosition(i + 1, j))
            {
               if (map.HasWay(i + 1, j) && m[i + 1, j] == 0)
               {
                  loop = true;
                  m[i + 1, j] = count + 1;
                  if (map.isCircle(i + 1, j))
                  {
                     loop = false;
                     found = true;
                     fountPoint = new Point(i + 1, j);
                     break;
                  }
               }
            }
            if (isOkPosition(i, j + 1))
            {
               if (map.HasWay(i, j + 1) && m[i, j + 1] == 0)
               {
                  loop = true;
```

```
            m[i, j + 1] = count + 1;
            if (map.isCircle(i, j + 1))
            {
                loop = false;
                found = true;
                fountPoint = new Point(i, j + 1);
                break;
            }
        }
    }
    if (isOkPosition(i - 1, j))
    {
        if (map.HasWay(i - 1, j) && m[i - 1, j] == 0)
        {
            loop = true;
            m[i - 1, j] = count + 1;
            if (map.isCircle(i - 1, j))
            {
                loop = false;
                found = true;
                fountPoint = new Point(i - 1, j);
                break;
            }
        }
    }
    if (isOkPosition(i, j - 1))
    {
        if (map.HasWay(i, j - 1) && m[i, j - 1] == 0)
        {
            loop = true;
            m[i, j - 1] = count + 1;
            if (map.isCircle(i, j - 1))
            {
                loop = false;
                found = true;
```

```
                    fountPoint = new Point(i, j - 1);
                    break;
                }
            }
        }
        if (isOkPosition(i - 1, j - 1))
        {
            if (map.HasWay(i - 1, j - 1) && m[i - 1, j - 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(-1, -1)))
            {
                loop = true;
                m[i - 1, j - 1] = count + 1;
                if (map.isCircle(i - 1, j - 1))
                {
                    loop = false;
                    found = true;
                    fountPoint = new Point(i - 1, j - 1);
                    break;
                }
            }
        }
        if (isOkPosition(i + 1, j + 1))
        {
            if (map.HasWay(i + 1, j + 1) && m[i + 1, j + 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(1, 1)))
            {
                loop = true;
                m[i + 1, j + 1] = count + 1;
                if (map.isCircle(i + 1, j + 1))
                {
                    loop = false;
                    found = true;
                    fountPoint = new Point(i + 1, j + 1);
                    break;
                }
```

```
                }
            }
            if (isOkPosition(i - 1, j + 1))
            {
                if (map.HasWay(i - 1, j + 1) && m[i - 1, j + 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(-1, 1)))
                {
                    loop = true;
                    m[i - 1, j + 1] = count + 1;
                    if (map.isCircle(i - 1, j + 1))
                    {
                        loop = false;
                        found = true;
                        fountPoint = new Point(i - 1, j + 1);
                        break;
                    }
                }
            }
            if (isOkPosition(i + 1, j - 1))
            {
                if (map.HasWay(i + 1, j - 1) && m[i + 1, j - 1] == 0 &&
!map.CheckCellsToConnection(new Point(i, j), new Point(1, -1)))
                {
                    loop = true;
                    m[i + 1, j - 1] = count + 1;
                    if (map.isCircle(i + 1, j - 1))
                    {
                        loop = false;
                        found = true;
                        fountPoint = new Point(i + 1, j - 1);
                        break;
                    }
                }
            }
```

```
                }
            }
            if (found)
                break;
        }
        count++;
    }
    Console.WriteLine();
    for (int i = 0; i < 40; i++)
    {
        for (int j = 0; j < 40; j++)
            Console.Write(m[i, j] + ", ");
        Console.WriteLine();
    }
    if (found)
    {
        Path pp = new Path(this);
        int i = fountPoint.X;
        int j = fountPoint.Y;
        pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2, j *
rectangle_size + (rectangle_size + 1) / 2));
        pp.setPointVal(i, j);
        while (m[i, j] > 1)
        {
            if (i > 0 && m[i - 1, j] == (m[i, j] - 1))
            {
                i = i - 1;
                pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2, j
* rectangle_size + (rectangle_size + 1) / 2));
                pp.setPointVal(i, j);
            }
            else
                if (j > 0 && m[i, j - 1] == (m[i, j] - 1))
                {
                    j = j - 1;
```

```
                    pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) / 2,
j * rectangle_size + (rectangle_size + 1) / 2));
                    pp.setPointVal(i, j);
                }
                else
                    if (m[i + 1, j + 1] == (m[i, j] - 1) )
                    {
                        i++;
                        j++;
                        pp.Points.Add(new Point(i * rectangle_size + (rectangle_size + 1) /
2, j * rectangle_size + (rectangle_size + 1) / 2));
                        pp.setPointVal(i, j);
                    }
                    else
                        if (i > 0 && j > 0 && m[i - 1, j - 1] == (m[i, j] - 1) )
                        {
                            i--;
                            j--;
                            pp.Points.Add(new Point(i * rectangle_size + (rectangle_size +
1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                            pp.setPointVal(i, j);
                        }
                        else
                            if (m[i + 1, j] == (m[i, j] - 1))
                            {
                                i++;
                                pp.Points.Add(new Point(i * rectangle_size + (rectangle_size
+ 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                                pp.setPointVal(i, j);
                            }
                            else
                                if (m[i, j + 1] == (m[i, j] - 1))
                                {
                                    j++;
```

```
                    pp.Points.Add(new Point(i * rectangle_size +
(rectangle_size + 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                    pp.setPointVal(i, j);
                }
                else
                    if (j > 0 && m[i + 1, j - 1] == (m[i, j] - 1))
                    {
                        i++;
                        j--;
                        pp.Points.Add(new Point(i * rectangle_size +
(rectangle_size + 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                        pp.setPointVal(i, j);
                    }
                    else
                        if (i > 0 && m[i - 1, j + 1] == (m[i, j] - 1))
                        {
                            i--;
                            j++;
                            pp.Points.Add(new Point(i * rectangle_size +
(rectangle_size + 1) / 2, j * rectangle_size + (rectangle_size + 1) / 2));
                            pp.setPointVal(i, j);
                        }
                }
                hasPossiblePath = true;
                lastPossiblePath = pp;
                return;
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            lastPossiblePath.DrawPath();
        }
    }
}
```

MAP.CS

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    class Map
    {
        public Cell[,] map;
        private int number_of_line;
        private List<Path> pathes;
        private int number_of_circle;
        public List<Circle> circles;
        Form1 form1;
        public Map(Form1 form)
        {
            form1 = form;
            map = new Cell[1000, 1000];
            for (int i = 0; i < 1000; i++)
                for (int j = 0; j < 1000; j++)
                    map[i, j] = new Cell();
            number_of_line = 999;
            number_of_circle = 0;
            pathes = new List<Path>();
            circles = new List<Circle>();
        }
        public void AddLine(Path path)
        {
            // lines add with index from 1000 to infinity
            number_of_line++;
            pathes.Add(path);
            for (int i = 0; i < (path.ArrayPoints.Count); i++)
```

```
                map[path.ArrayPoints[i].X,
path.ArrayPoints[i].Y].AddPath(number_of_line, i);
            int ii = path.ArrayPoints[(path.ArrayPoints.Count - 1) / 2].X;
            int jj = path.ArrayPoints[(path.ArrayPoints.Count - 1) / 2].Y;
            Circle c = new Circle(form1, form1.circle_raduis, ii * form1.rectangle_size, jj
* form1.rectangle_size);
            c.DrawCircle();
            c.hand = 2;
            AddCircle(c, form1.rectangle_size);
        }
        public void AddCircle(Circle c, int rectangle_size)
        {
            // circles add index from 1 to 999
            number_of_circle++;
            int i = c.getLocation().X / rectangle_size;
            int j = c.getLocation().Y / rectangle_size;
            map[i, j].AddCircle(c);
            circles.Add(c);
        }
        public bool isCircle(int i, int j)
        {
            return map[i, j].HasCircle();
        }
        public Circle getCircleFromMap(int i, int j)
        {
            return map[i, j].GetCircle();
        }
        public void Redraw()
        {
            for (int i = 0; i < circles.Count; i++)
                circles[i].DrawCircle();
            for (int i = 0; i < pathes.Count; i++)
            {
                pathes[i].DrawPath();
            }
```

```
        }
        public bool HasWay(int x, int y)
        {
            return map[x, y].HasWay();
        }
        public bool CheckCellsToConnection(Point p, Point move)
        {
            //map[p.X + move.X, p.Y];
            //map[p.X, p.Y + move.Y];
            //Console.WriteLine(p.X + ", " + p.Y);
            if ((p.X + move.X) < 0) return true;
            if ((p.Y + move.Y) < 0) return true;

            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path0, map[p.X,
p.Y + move.Y].path0))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path0, map[p.X,
p.Y + move.Y].path1))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path0, map[p.X,
p.Y + move.Y].path2))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path1, map[p.X,
p.Y + move.Y].path0))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path1, map[p.X,
p.Y + move.Y].path1))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path1, map[p.X,
p.Y + move.Y].path2))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path2, map[p.X,
p.Y + move.Y].path0))
                return true;
```

```
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path2, map[p.X,
p.Y + move.Y].path1))
                return true;
            if (CheckTwoCellsForConnection(map[p.X + move.X, p.Y].path2, map[p.X,
p.Y + move.Y].path2))
                return true;
            return false;
        }
        public bool CheckTwoCellsForConnection(Point p1, Point p2)
        {
            if (p1.X == p2.X && (Math.Abs(p1.Y - p2.Y) == 1))
                return true;
            return false;
        }
        public List<Point> GetPossibleCircles()
        {
            List<Point> list = new List<Point>();
            for (int i = 0; i < 1000; i++)
                for (int j = 0; j < 1000; j++)
                {
                    if (map[i, j].HasCircle() && map[i,j].HasWay())
                    {
                        list.Add(new Point(i,j));
                    }
                }
            return list;
        }
        public Cell[,] getMap()
        {
            return map;
        }
    }
}
```

PATH.CS

```
using System;
```

```
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    class Path
    {
        public int[,] path;
        private int path_length;
        public List<Point> Points;
        public List<Point> ArrayPoints;
        public Point FirstPoint;
        public Point LastPoint;
        private Graphics objGraphics;
        private Pen pen = Pens.Red;

        public void Clone(Path p)
        {
            this.path = p.path;
            this.path_length = p.path_length;
            this.Points = p.Points;
            this.ArrayPoints = p.ArrayPoints;
            this.FirstPoint = p.FirstPoint;
            this.LastPoint = p.LastPoint;
            this.objGraphics = p.objGraphics;
        }
        public Path(Form1 form)
        {
            Points = new List<Point>();
            ArrayPoints = new List<Point>();
            path = new int[100, 100];
            path_length = 0;
```

```
            objGraphics = form.objGraphics;
        }
        public void Reset()
        {
            Points = new List<Point>();
            ArrayPoints = new List<Point>();
            path = new int[1000, 1000];
            path_length = 0;
        }
        public bool SelfCheck(int x, int y)
        {
            if (x < 0 || y < 0)
                return false;
            if (path_length == 0) return true;
            Point p = new Point(x - LastPoint.X, y - LastPoint.Y);
            //Console.WriteLine(p.X+" , "+p.Y);
            if (Math.Abs(p.X + p.Y) == 1 && p.X > -2 && p.X < 2 && p.Y < 2 && p.Y > -
2) return true;
            //dioganal harakat
            if (Math.Abs(p.X) == 1 && Math.Abs(p.Y) == 1)
                return Math.Abs(path[LastPoint.X + p.X, LastPoint.Y] - path[LastPoint.X,
LastPoint.Y + p.Y]) != 1;

            return false;
        }
        public bool CheckWithMap(int x, int y, Map map)
        {
            if (path_length < 2)
                if (map.isCircle(x, y)) return true;
            Point p = new Point(x - LastPoint.X, y - LastPoint.Y);
            //dioganal harakat
            if (x > 0 && y > 0)
                if (map.isCircle(LastPoint.X, LastPoint.Y) && path_length > 1) return
false;
```

```
            if (Math.Abs(p.X) == 1 && Math.Abs(p.Y) == 1)
                return !map.CheckCellsToConnection(LastPoint, p) && map.HasWay(x,
y);

            return map.HasWay(x, y);
        }
        public bool checkPointForZero(int i, int j)
        {
            if (i < 0 || j < 0) return false;
            return this.path[i, j] == 0;
        }
        public bool checkPoint(int x, int y, Map map)
        {
            if (!map.isCircle(LastPoint.X, LastPoint.Y) && FirstPoint.X == x &&
FirstPoint.Y == y && path_length > 2)
            {
                if (map.map[x, y].circle.hand < 3)
                    return true;
                else
                    return false;
            }
            return checkPointForZero(x, y) && SelfCheck(x, y) && CheckWithMap(x, y,
map);
        }
        public void setPointVal(int i, int j)
        {
            ArrayPoints.Add(new Point(i, j));
            if (path_length == 0)
                FirstPoint = new Point(i, j);
            path_length++;
            path[i, j] = path_length;
            LastPoint = new Point(i, j);
        }
        public bool CanAddLine(Map map)
        {
```

```
            bool stat = true;
            if (!map.isCircle(FirstPoint.X, FirstPoint.Y))
            {
               stat = false;
            }
            else
            {
               if (map.getCircleFromMap(FirstPoint.X, FirstPoint.Y).hand > 3)
                  stat = false;
            }
            if (!map.isCircle(LastPoint.X, LastPoint.Y))
            {
               stat = false;
            }
            else
            {
               if (map.getCircleFromMap(LastPoint.X, LastPoint.Y).hand > 3)
                  stat = false;
            }
            if (LastPoint == FirstPoint)
            {
               if (stat)
                  if (map.getCircleFromMap(LastPoint.X, LastPoint.Y).hand > 2)
                     stat = false;
            }
            return stat;
         }
         public bool CloseSession(int rectangle_size, Map map, Graphics objGraphics,
Pen p = null)
         {
            if (p != null)
               pen = p;
            else
               pen = Pens.Red;
            bool ans = false;
```

```
            if (Points.Count < 2)
                return ans;
            if (CanAddLine(map))
            {
                map.map[FirstPoint.X, FirstPoint.Y].setCircleHand();
                map.map[LastPoint.X, LastPoint.Y].setCircleHand();
                ans = true;
                map.AddLine(this);
            }
            objGraphics.Clear(Color.White);
            map.Redraw();
            return ans;
        }
        public int Length()
        {
            return path_length;
        }
        public void DrawPath()
        {
            if (Points.Count < 2)
                return;
            objGraphics.DrawCurve(pen, Points.ToArray());
        }
        public Path getThis()
        {
            return this;
        }
    }
}
```

ЗАКЛЮЧЕНИЕ

Термин топологическая игра был впервые введен Берге [2], который определил основные идеи и формализм по аналогии с топологическими группами. Оказывается, что некоторые фундаментальные топологические конструкции имеют естественный аналог в топологических играх, примеры из них свойство Бэра, бэровские пространства, полнота и свойства сходимости, свойства разделения, покрытия, непрерывные образы, множества Суслина, и сингулярное разложение. В то же время, некоторые топологические свойства, которые возникают естественным образом в топологических играх могут быть обобщены за пределы теоретика - игровой контекст в силу этой двойственности, топологические игры широко используются для описания новых свойств топологических пространств.

«Недавно мой друг, изучающий классическую филологию в Кембриджском университете, познакомил меня с игрой «Рассада». В течение последнего семестра на этой игре буквально помешался весь Кембридж. Некоторые особенности игры не лишены интереса с точки зрения топологии». Так начиналось письмо, которое Мартин Гарднер получил в апреле 1967 года от одного студента-математика из Англии. Вскоре начали поступать и другие сообщения о том, что «Рассада» привилась и пышно расцвела на благодатной почве Кембриджа.

Теория топологических игр относится к наукам геометрии и топологии. В данной диссертации рассмотрены вопросы касающееся топологических игр «Рассада» и «Брюссельская капуста». Проблема

исследование решается теорией топологических игр и методами топологии. В ходе исследования рассмотрены такие вопросы, как изучение топологических игр Рассада и Брюссельская капуста, обобщение топологических игр Рассада и Брюссельскяя капуста, исследование топологических игр Рассада и Брюссельскоя капуста на поверхности тора, создание программное обеспечение игр Рассада и Брюссельская капуста.

В первой главе рассмотрены топологичекие игры “Рассада” и “Брюссельская капуста” на плоскости, их истории, основные определения. Основная задача этой главы расрыт в третьем параграфе, в котором детально изложены теория обобщенной игры “Рассада и брюссельская капуста”.

Во второй главе игра “Брюссельская капуста” изучен на поверхности тора, также рассматривается обобщенная игра “Брюссельская капуста” на поверхности тора.

Третья глава называется “Программное обеспечение игр “Рассада и брюссельская капуста”. В нем приводится программное обеспечение игры “Рассада” и программное обеспечение игры “ Брюссельская капуста”

Данная магистрская диссертация служит развитию теории топологических игр и её применение.

Литература

1. Мирзиёев Ш.М. Мы все вместе построим свободное, демократическое и процветающее государство Ўзбекистан. - Т.: “Ўзбекистан”, 2016. – 40 с.
2. Berge C. Topological games with perfect information. Contributions to the theory of games, vol. 3, 165–178. Annals of Mathematics Studies, no. 39.Princeton University Press, Princeton, N. J., 1957.
3. Gardner M. Mathematical Games, Scientific Amer. 197 (1957).
4. Berlekamp E., Conway J., Guy R.Winning Ways for your Mathematical Plays, Vol. I and II, Academic Press, (1982).
5. Delucchi E., Gaiffi G., Pernazza L. Giochi e percorsi matematici, Springer, 2012.
6. Гарднер М. Математические новеллы. Москва. 1974.
7. Telgarsky R. Topological games and analytic sets, Houston J. Math. 3 (1977), 549-553.
8. Маматов М.Ш., Жабборов М.М. Топологические игры двух лиц о разделении пространств рассада и брюссельская капуста. “Актуальные проблемы гуманитарных и естественных наук ”. Москва. - 2016.- №5.- С. 16-19.
9. Ulam S.M. Combinatorial analysts in infinite sets and some physical theories, SIAM Rev. 6(1964) 343-355.
10. Кобаяси Ш., Номидзу К. Основы дифференциальной геометрии. т.1.М.: 1981. 344 с.

11. Lachlan A.L. On some games which are relevant to the theory of recursively enumerable sets, Ann. of Math. 91 (1970), 291-310.

12. Martin Baxter, Unfair games, *Eureka* **50** (1990), 60–68.

13. Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy, *Winning ways for your mathematical plays. Vol. 1*, second ed., A K Peters Ltd., 2001.

14. Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy, *Winning ways for your mathematical plays. Vol. 3*, second ed., A K Peters Ltd., 2003.

15. Grant Cairns and Korrakot Chartarrayawadee, Brussels sprouts and cloves, *Math. Mag.* **80** (2007), no. 1, 46–58.

16. D'Alarcao H. and Moore T. E. Euler's formula and a game of Conway's, *J.Recreational Math.* **9**(1977), 249–251.

17. Маматов М.Ш., Жабборов М.М. Топологические игры двух лиц о разделении пространств рассада и брюссельская капуста. Актуальнқе вопрос анализа/ труд респ. Конф.-Карши, Узбекистан, 2016.-С. 289-292.

18. Applegate D., Jacobson G., and Sleator D. Computer Analysis of Sprouts, Tech. Report CMU-CS-91-144, Carnegie Mellon University Computer Science Technical Report, 1991.

19. George K. Francis and Jeffrey R.Weeks, Conway's zip proof, American Mathematical Monthly 106 (1999), 393–399.

20. Martin Gardner, Mathematical games: of sprouts and brussels sprouts, games with a topo-logical flavor, Scientific American 217 (July 1967), 112–115.

21. Julien Lemoine and Simon Viennot, A further computer analysis of sprouts, http://download.tuxfamily.org/sprouts/sprouts-lemoine-viennot-070407.pdf, 2007.

22. Cooper M. Graph Theory and the Game of Sprouts, *American MathematicalMonthly*, **100**(May): 478-482.

23. Gardner M. *Mathematical Carnival*, chapter 1, pages 3-11, Alfred A.Knopf, New York, 1975.

24. Lam T.K. Connected Sprouts, *American Mathematical Monthly*, **104**(February):116-119.

25. Applegate D., Jacobson G. and Sleator D. Computer analysis of Sprouts, in The mathemagician and pied puzzler, Elwyn Berlekamp and Tom Rodgers (eds.), A K Peters Ltd., Natick,MA, 1999.

26. Martin Gardner, The colossal book of mathematics, W. W. Norton & Co. Inc., New York, 2001.

27. Richard K. Guy, Graphs and games, in Selected topics in graph theory, 2, pp. 269–295, Academic Press, London, 1983.

28. Маматов М.Ш., Жабборов М.М. Топологические игры двух лиц о разделении пространств, тезисы проблемы современной топологии и её приложения/ международная конференция.-Ташкент, Узбекистан 2016.-С .218-220.

29. Gordon D. Prichett, The game of Sprouts, Two-Year College Math. J., 7 (1976), no. 4, 21–25.

30. Danny Purvis, World Game of Sprouts Association, http://www.geocities.com/chessdp/.

31. Зейферт Г., Трельфалль В. Топология. Москва. 2001.